freelance_Project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

project28 8007592194

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48		React+Springboot+MySql
49		React+Springboot+MySql
50		React+Springboot+MySql

@freelance_project28 8007592194

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/cceesept2023

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project

Prasanna wadekar 9970248407 36=00

Core Java Notes

Day 01 (5 hours)

=======

+ Java Books:

- Core Java Vol 1 & 2 - Hoarsman
- SCJP Programmer's guide - Khalid Mughal
- Java Programming Language - Jame Gonsling
- Inside JVM -

+ Java Programming Language:

- Invented by "James Gosling"
- Company "Sun Microsystem" -> "Oracle Inc".
- Originally Java was invented for embedded systems. In fact first product developed using Java technology was a "Remote Control" named as "*7".
- However in spite lot of efforts from Java team, Java was not considered by industry at that time.
- In 1990's people started using internet. However internet support was limited to static web pages.
- Around 1995 Java use applets as interactive java programs that can be executed on client browsers. In short span of time, applets and hence Java become popular for internet.
- There are three main streams were developed using Java language:

1. Java SE:
    - Desktop based java applications (Console & GUI applications).
    - Core language part of Java i.e. OOP and other features.
    - Versions pulished:
        - Java 1.0
        - Java 1.1
        - J2SE 1.2 -> Added collection framework
        - J2SE 1.3
        - J2SE 1.4 -> another stable and popular Java
        - JavaSE 5.0 -> Added generics, annotations,
        - JavaSE 6
        - JavaSE 7 -> Added Parallel programming,...
        - JavaSE 8
2. Java EE:
    - Web based / Enterprise application
    - Servlets, JSP, EJB, Struts, Spring
    - Versions published:
        - J2EE 1.3
        - J2EE 1.4

PRASANNA

- JavaEE 5.0
- JavaEE 6
- JavaEE 7

3. Java ME:
   - Java for embedded devices.

+ C/C++ Compilation & Execution:

[.c/.cpp] -> Compiler -> [.obj] -> Linker -> [.exe]

Source Code	Object Code	Executable
	Machine Instru	Machine Instr

|
Operating Sys
|
Processor (HW)

+ Java Compilation & Execution:

Hello.java

```
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- **COMPILATION COMMAND: javac Hello.java**
  - Name of .java file (with extension)
  - .class file is always created as per class-name (not .java file name).
  - If single .java file contains multiple java classes, then multiple .class files will be created.
- **EXECUTION COMMAND: java Hello**
  - Name of .class file (without extension)
  - It is expected that in this class, main() is prsent

[Hello.java] -> Compiler -> [Hello.class]

Source code	Java Executable
Java Code	Byte Code

|

```
JVM
 |
[Machine Instr]
      |
Processor(HW)
```

+ JVM, JRE and JDK:

- Java Virtual Machine:
  - JVM is hypothetical machine assumed by Java compiler and is emulated in software as part of JRE.
  - Machine level instructions of JVM are of one byte each instruction and hence it is called as "byte code".
  - java ClassName
  - Here "java" is application launcher.

  - JVM Architecture:

```
                    |-------------------|
.class -->     |    Class Loader    |
                    |-------------------|
                    | Byte Code Verifier |
                    |-------------------|
                    |          JIT          | --> Processor(Hw)
                    |-------------------|
                    | Garbage Collector |
                    |-------------------|
```

- Java Runtime Environment:

- JRE is required for execution of java programs (.class) files.
- JRE = JVM + java tools + java libraries (.jar)
- JRE can be downloaded from oracle site for particular OS platform. In other words, JRE is platform depenedent.

- Java Development Kit:

- JDK is required for developing java applications.
- JDK = JRE + java dev tools (compiler, debugger,...)
- JDK can be downloaded from oracle site for particular OS platform. In other words, JDK is platform depenedent.

+ Java Features:

- Portable:
- Compile Once and Run Anywhere
- Java code is compiled into .class files. These .class files can be executed on any OS for which JRE is present.
- Even though same .class can run on many platforms (OS), for certain features behaviour will be OS dependent e.g. File IO, Threads, Sockets, AWT, ...
    - Thus java is not purely "Platform Independent".
- Architecture Neutral:
    - JVM (JIT) convert byte code into native code as per underlying CPU architecture.
    - So same java code can be executed on any CPU arch for which JRE is present.

+ Reading work:

- Core Java Vol 1
    - Java History
    - Java Buzzwords

+ Java Code compilation and Execution:

- PATH is an Ennvironment Variable of the OS. It contains list of directories seperated by semicolon ; (on windows) or colon : (on UNIX/Linux).
- When any program is executed on command line without giving its full path, OS will search it into all directories mentioned in PATH variable.
- If you want to execute any program without giving its full path, you must add its directory path into PATH variable.

- To see current PATH variable:

[WIN] set PATH

[LINUX] echo $PATH

- To add a dir path in in PATH variable:

[WIN] set PATH=new_dir_path;%PATH%

[LINUX] export PATH=new_dir_path:$PATH

- To permantly set PATH variable:

[WIN] My Computer -> Properties -> Advanced
-> Environment Variable -> Select PATH -> click EDIT -> add new_dir_path follwed by ; (do not remove/overwrite existing PATH).

[LINUX] in .bashrc file (in your home dir) add cmd:
export PATH=new_dir_path:$PATH

- To unset/nullify PATH variable:

[WIN] set PATH=

[LINUX] export PATH=

- **To add path of java compiler (javac) into PATH var:**

  [WIN] set PATH=C:\Program Files\Java\jdk1.xyz\bin;%PATH%

  [LINUX] export PATH=/usr/java/jdk1.xyz/bin:$PATH

- **javac -d dirpath FileName.java**
  - In this example, given Java file will be compiled and .class file(s) will be created in the give dirpath.
  - e.g. javac -d . Hello.java
    -> will create Hello.class in current directory.
  - e.g. javac -d D:\temp Hi.java
    -> will create Hi.class in D:\temp directory.

- **java ClassName**
  - This will start java application laucher (java.exe), load JVM in it and it will execute given .class file.
  - After loading and verifying class, execution begin from main() method.
  - If proper main() is not found, JVM will throw error.

+ Hello world example:

```
import java.lang.System;
class Hello {
	public static void main(String[] args) {
		System.out.println("Hello, world!");
	}
}
```

- In java, global data/methods are not allowed. Even main() method must be declared within some class.
- When program exxecution begins, JVM calls/executes main() method.
- Here we expect that main() method should be invoked from outside the class (JVM) and hence we declare it as public.
- JVM invoke main() method directly without creating object of the class and hence main() should be declared as static.
- main() does not return any value, so its return type is void.
- main() method can take command line arguments and hence it has an argument : String[] args
- System.out.println("Hello, world!");
  - Here "System" is a pre-defined class
  - "out" is public static member of that class.
  - "out" is of type of "PrintStream" class.

- println() is a method in "PrintStream", which is used to print a given message on the screen followed by a newline character.

- System.in encapsulates stdin.

- System.out encapsulates stdout.

- System.err encapsulate stderr.

- System.out.printf() can also be used to print formatted output on the screen just like C language printf() function. Supported format specifiers are %d, %f, %s, %c.

+ Java primitive types:

- Primitive types are not classes in java to improve the performance.
- Integer types:
  - byte (1 byte) - unsigned : 0 to 2^8-1
  - short (2 byte) - signed : -2^15 to +2^15-1
  - int (4 byte) - signed : -2^31 to +2^31-1
  - long (8 byte) - signed : -2^63 to +2^63-1
- Real (numbers) types:
  - float (4 bytes)
  - double (8 bytes)
- character types:
  - char (2 bytes) : unicode characters : 'A'
  - * String- set of characters
    - "Hello" - string const or string literal
- boolean type:
  - boolean (1 bit) : true or false

- Since java is strictly type-checking only a few conversions are allowed (not all).
- Conversion from boolean to integer type and vice-versa is not supported directly.
- Conversion from char to integer type and vice-versa is not supported directly.
- Conversion from narrow data type to wider data type does not need type-casting. However conversion from wider data type to narrow data type needs type-casting.

```
short a = 10;
int b = a;          // OKAY
int x = 10;
short y = x;        // ERROR
short z = (short)x;     // ALLOWED
```

+ Java Classes and Objects:

- Class is user-defined data type.
- Class contains fields and methods.
- Variables of class type are called as "Objects".
- In java, objects can be created only on heap using "new operator". In other words, objects cannot be created on stack.

```
- public static void main(String[] args) {
        Person p1 = new Person();
        // ...
}
```

- "p1" is a reference, keeping address of Person object created on heap.
- In java reference can be "null" or can be initialized after declaration.
- If local reference is used before initialization, compiler will raise error.

```
Person p1;
p1.display();    // error
```

- Accessing members on null reference will raise "NullPointerException" (at runtime).

```
Person p1=null;
p1.display();    // exception
```

- **Methods:**
    - **Constructor:**
    - Same name as that of class, no return type.
    - Used to initialize fields.
    - If not implemented, each class will have a default paramterless constructor given by the compiler.
        - Parameterless constructor
        - Parameterized constructor

```
Person() {
        this("", "", 0); // call to param ctor
}
Person(String name, String address, int age) {
        // ...
}
```

- In java, one constructor of class can be accessed from another constructor of that class using "this" keyword.
- However, such constructor call must be first line of the constructor; otherwise compiler raise error.
    - In java, if fields are not initialized:
- Primitive type variables will be initialized to default value e.g. int=0, boolean=false, etc.

- reference variables will be initialized to null.

- In java, fields can be initialized at the point of declaration in the class.

```
class Person {
        String name = "nitin";
        int age = 40;
        // ...
}
```

- Object initializer syntax/block:

```
class Person {
        String name;
        int age;

        // object initializer block
        {
                this.name = "something";
                this.age = 10;
        }

        // ...
}
```

- When a new object is created, all values initialized at point of declaration of fields will be assigned.
- Then object initializer block(s) will be executed in the order of their declaration in the class.
- Finally constructor is executed as per creation of the object.

```
p1 = new Person(); // paramless
p2 = new Person("","",2); // parameterized
```

* Java uses Hungarian notation for class names, while camel notation for field/method names.

- getters/inspectors:

```
int getAge() {
        return this.age;
}
```

- setters/mutators:

```
void setAge(int age) {
```

```
                this.age = age;
            }
```

- facilitators:

```
            display() / accept()
```

+ **User input:**

- Using "Scanner" class one can get input from end user.
- This class is defined in "java.util" package (not in default "java.lang" package).
- Hence before using this class, importing package is mandetory (as shown in example).

- import java.util.Scanner;

```
class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.printf("Enter full name : ");
        String name = sc.nextLine();
        System.out.printf("Enter name : ");
        String name = sc.next();
        System.out.printf("Enter roll : ");
        int roll = sc.nextInt();
        System.out.printf("Enter marks : ");
        double marks = sc.nextDouble();
        sc.nextLine(); // flush \n char
        System.out.printf("Enter School name : ");
        String school = sc.nextLine();
        // ...
        sc.close();
    }
}
```

Day 02 (5 hours)

==========

+ **Access Modifiers:**

- Class members can have one of the following access modifier:
    1. public
    2. protected
    3. private

4. default (package)

- Usually fields are declared as private and methods are declared public.

```
- class Person {
        private String name;
        private int age;
        public Person() {
                // ...
        }
        public void display() {
                // ...
        }
}
```

+ Compilation and Execution Standard Practice:

```
[project]
        |- [src]
        |       |- *.java
        |- [classes]
                |- *.class
```

0. Create directory structure as above & save .java files into "src" directory.
1. Open command prompt and go to "src" directory.
2. CMD> javac -d ..\classes Main.java
3. CMD> cd ..\classes
4. CMD> java Main

+ static keyword in java:

- Static members do not belong to the object, they belong to the class.
- Static members can be accessed directly by using class name and dot (.) operator.
- Hence to access static members there is no need to create object of the class.
- However, Like C++, java allows access to the static members using dot operator on objects.

+ static fields

- Static fields do not contribute to size of object.
- The data that is common for all objects of the class should be declared as static.

+ static methods

- Static methods do not receive implicit "this" pointer and hence they cannot access non-static members of the class directly.

- Static methods can access static members of the class. .
- Static methods are written to manipulate static fields in the class.
- Static methods are sometimes implemented as entry-point to the class. In other words, they are used to create objects of class.
    - e.g. Calendar cal = Calendar.getInstance();
        - Creating object using such static method in the class is called as "static factory pattern".

```
+ static block
    - class Sample {
        // ...
        static {
            // ...
        }
        // ...
    }
```

- When class is loaded first time into JVM. class-loader invokes static block(s) declared in the class.
- There can be one or more number of static blocks declared in the class.
- When class is accessed first time, all these static blocks will be invoked in the order of their declaration in the class.
- Static blocks are used to perform one-time initialization for the "class".
- Initialization of static members when class is accessed first time.
- JDBC driver registration must be done once before establishing database connection.

e.g. **Singleton class:**

```
- class Singleton {
    static Singleton obj = null;
    public static Singleton getInstance() {
        if(obj==null)
            obj = new Singleton();
        return obj;
    }
    // fields
    private Singleton() {
    }
    // methods
}

- class Singleton {
```

```
        static Singleton obj = null;
        static {
            obj = new Singleton();
        }
        public static Singleton getInstance() {
            return obj;
        }
        // fields
        private Singleton() {
        }
        // methods
    }
    class Main {
        public static void main(String[] args) {
            Singleton s1 = Singleton.getInstance();
            Singleton s2 = Singleton.getInstance();
            // ...
        }
    }
    + static member class
        - Will be discussed with "inner classes".

+ Composition:
    - Represents "has-a" relationship between the objects.
    - class Date {
        private int day, month, year;
        // ...
    }
    - class Book {
        private String name, author;
        private double price;
        private int pages;
        private Date publish;
        public Book() {
            this.name = "";
            this.author = "";
            this.publish = new Date();
```

```
        }
        public Book(String name, String author, double price, int pages, int
                day, int month, int year) {
            this.name = name;
            this.author = author;
            this.price = price;
            this.pages = pages;
            this.publish = new Date(day,month,year);
        }
        public Book(String name, String author, double price, int pages,
                                                    Date publish) {
            this.name = name;
            this.author = author;
            this.price = price;
            this.pages = pages;
            this.publish = publish;
        }
        public void display() {
            System.out.printf("%s\n%s\n", name, author);
            System.out.printf("%f\n%d\n", price, pages);
            publish.display();
        }
    }
    class Main {
        public static void main(String[] args) {
            Book b1 = new Book();
            b1.display();

            Book b2 = new Book("book1", "au1", 100.00, 30, 1, 1, 2014);
            b2.display();

            Date d = new Date(2,2,2014);
            Book b3 = new Book("bk2", "au2", 200.00, 40, d);
            b3.display();
            Book b4 = new Book("bk4", "au4", 20.0, 10, new Date(3,32014));
            b4.display();
        }
    }
```

+ **Arrays:**

- In java, arrays are objects. So they must be created on heap using "new" operator.
- Syntax:
  - `int[] arr;`
    - Just declares a reference to array of ints.
  - `int[] arr = new int[5];`
    - By default array elements are initialized to zero.
  - `int[] arr = new int[5] { 11, 22, 33, 44, 55 };`
    - Initializing array elements at point of declaration.
- Internally array is object of "java.lang.Array" class and it holds array elements and also a "length" field to represent number of elements in the array.
- To access array elements:

```
-    for(i=0; i < arr.length; i++)
         System.out.printf("%d\n", arr[i]);
-    for(int n : arr)
         System.out.printf("%d\n", n);
```

- If you try to access array element at invalid index (index < 0 or index >= length), then exception will be thrown.

- When we create array of objects, internally array of references is created. By default each reference is "null".
- Programmer must initialized each reference to a valid object to avoid "NullPointerException".

```
-    Date[] arr = new Date[5];
     arr[0] = new Date(1,1,2000);
     arr[1] = new Date(1,1,2001);
     arr[2] = new Date(1,1,2002);
     arr[3] = new Date(1,1,2003);
     arr[4] = new Date(1,1,2004);
     for(int i=0; i<arr.length; i++)
         arr[i].display();
```

- Each java program receives a set of command line arguments given by user while executing program on command line, in form of an array argument to main() function.
- Unlike C/C++, 0th element of the array is not name of executable / .class file.
- Main.java -> class Main ->

```
public static void main(String[] args) {
    for(int i=0; i<args.length; i++)
        System.out.printf("%s\n", args[i]);
}
```

- If program is executed like:

  java Main Pune Karad Satara Sangli

- In this case, "args" will be array of 4 elements

  0 -> Pune

  1 -> Karad

  2 -> Satara

  3 -> Sangli

+ Ragged Arrays:

  - In java, we can create array of array objects. Size of each array object may or may not be same.

```
for(int i=0; i<arr.length; i++) {
    for(int j=0; j<arr[i].length; j++) {
        System.out.printf("%d ", arr[i][j]);
    }
    System.out.println();
}
```

+ Java Method Arguments:

  - In java primitive types are always passed by value.

```
public static void swap(int a, int b) {
    int t = a;
    a = b;
    b = t;
    System.out.printf("a:%d, b:%d\n", a, b);//20 10
}
public static void main(String[] args) {
    int x=10, y=20;
    System.out.printf("x:%d, y:%d\n", x, y);//10 20
    swap(x, y);
    System.out.printf("x:%d, y:%d\n", x, y);//10 20
}
```

  - In java objects are always passed by reference.

```
public static void change(Date d) {
    d.setDay(24);
}
public static void main(String[] args) {
    Date d1 = new Date(12, 6, 1980);
```

```
        d1.display();    // 12/6/1980
        change(d1);
        d1.display();    // 24/6/1980
    }
```

- Variable Arity Method:

```
    public static int add(int... arr) {
        int sum=0;
        for(int i=0; i<arr.length; i++)
            sum += arr[i];
        return sum;
    }
```

- Above declared function can take variable number of integer arguments.
- add(1,2);
- add(1,2,3,4);
- Internally compiler creates an array of all arguments passed to the method and that array is passed as an argument to method.
- We can also pass array of ints to this method.

- To pass variable number of args of any data type the method should be declared as :

```
void method(Object... arr) {
    // ...
}
```

- Since "java.lang.Object" is base/super class for all java classes, it can hold objects of any type.

+ Inheritance:

- Inheritance represent "is-a" relationship.
- All members of "super" class are inherited to the "sub" class. Sub-class can have its own additional members.
- **Inheritance Types:**
  - Single
  - Multiple
  - Java does not support "multiple implementation inheritance". In other words, a class cannot be inherited from multiple super classes.
  - Java does support "multiple interface inheritance". In other words, a class/interface can be inherited from multiple "interfaces".
  - multi-level
  - heirarchial
- **Inheritance modes:**

- Java does not support mode of inheritance like C++.
- Inheritance in java is similar to "public" mode of inheritance in C++.

- **Syntax:**

```
class Person {
    private String name, address;
    private int age;
    public Person() {
        this.name = "";
        this.address = "";
        this.age = 0;
    }
    public Person(String name, String address, int age) {
        this.name = name;
        this.address = address;
        this.age = age;
    }
    // getters/setters
    // input() / display()
}
class Employee extends Person {
    private int empno;
    private double salary;
    public Employee() {
        this.empno = 0;
        this.salary = 0.0;
    }
    public Employee(String name, String address, int age, int
     empno, double salary) {
        super(name, address, age);
        this.empno = empno;
        this.salary = salary;
    }
    // getter/setter
    // input
    public void display() {
        super.display();
        System.out.printf("%d\n%d\n", empno, salary);
```

```
        }
    }
    class Main {
        public static void main(String[] args) {
            Person p1 = new Person("p1", "a1", 20);
            p1.display();

        Employee e2 = new Employee("e2", "a2", 30, 1001, 40000.00);
            e2.display();
        }
    }
```

Day 03 (2 hours)

=======

+ Inheritance:

- In java "sub-class" "extends" functionality of "super-class".
- All public and protected members of super-class are directly accessible from "sub-class" methods.
- However if name of member is super-class and sub-class is same, then super-class member can be accessed from sub-class method using "super" keyword.

```
    - e.g. Employee class
        public void display() {
            super.display();
            System.out.printf("%d\n%f\n", empno, salary);
        }
```

- Whenever object of sub-class is created, first super-class constructor is executed and then sub-class constructor is executed.
- If not specified, from sub-class constructor paramless constructor of super-class is called.
- We can invoke parameterized constructor of super-class from sub-class using "super" keyword.

```
- e.g. Employee class:
        public Employee(String name, String address, int age, int empno,
double salary) {
            super(name, address, age);
            this.empno = empno;
            this.salary = salary;
        }
```

- Super-class constructor can be invoked from the first line of sub-class constructor. Otherwise compiler raise error.

+ Object Slicing:

- When sub-class object is assigned to a super-class reference, using that reference we can access only super-class methods. Accessing sub-class methods on such reference will raise compile time error.

- e.g.

```
Person p1 = new Employee();
p1.setName("Nilesh"); // Allowed
p1.setSalary(30000.00);     // Error
```

+ Method Overriding:

- However if a method is redefined in the sub-class with the same signature as that of super-class; it is called as "method overriding".

- Unlike C++, java doesn't need "virtual" keyword; as each method in java is by default treated like virtual.

- In such case, method is always invoked depending on type of object (not depending on type of reference). This is called as "dynamic method dispatch".

```
- e.g.
    class Person     -> public void display() {...}
    class Employee   -> public void display() {...}
    Person p1 = new Employee();
    p1.display();    // Employee's display() is called
```

- While overriding a method, its access specifier can be widen in sub-class. Reverse is not allowed (compile time error).
- In other words, if method is "public" in super-class it must be overridden in sub-class as "public". Attempt to make it protected or private in sub-class raise compile time error.
- While overriding method in sub-class, return-type of method can remain same or can be of sub-class type (of return type).

+ Package:

- Package is collection of sub-packages, classes and other types.
- Package is a physical container (i.e. directory) containing sub-packages, classes and other types.
- A type/class can be added into the package by giving package declaration on the first line of .java file.

  package pkg1;

- Packages are used to avoid name-clashing (just like namespaces in C++).
- Class access modifiers:

  1. default:

     - Classes are accessible into the same package.

  2. public:

     - Classes can be accessed in the same package as well as outside the package.
     - To access a class outside the package, use import statment into that .java file.

- import java.util.Scanner;
- The one or more "import" statements can be written after "package" statement in the .java file (if present).
- One .java file can contain one or more java classes; however only one class can be "public" (in a .java file).
- Name of "public" class must match with .java file name. Otherwise compiler will raise error.

+ **Example of packages:**

- Person.java

```
==============
package p1;
public class Person {
	// ...
}
```

- Main.java

```
==============
import p1.Person;
//import p1.*;
public class Main {
	public static void main(String[] args) {
		Person p1 = new Person();
		// ...
	}
}
```

+ javac -d . Person.java
  - create "p1" package (dir) and Person.class file in that dir.
+ javac -d . Main.java
  - create Main.class in cur directory.
+ java Main
  - will execute program.

- **import p1.*;**
  - This will allow to access all public classes in "p1" package into the .java file (in which above line is written).
  - This will not allow classes under sub-packages of "p1".
  - However it is recommended practice to import each class individually as required.
  - A class without any package cannot be accessed in class with some package.

- **Nested packages:**

- It is possible to have package within another package. In fact, it is recommended practice to have hierarchial package names.

- Example:

Person.java

==========

```
package com.sunbeam.wimc.pkg1;
public class Person {
	// ...
}

Main.java
==========
package com.sunbeam.wimc.pkg2;
import com.sunbeam.wimc.pkg1.Person;
public class Main {
	// ...
}
```

CMD> javac -d . Person.java

CMD> javac -d . Main.java

CMD> java com.sunbeam.wimc.pkg2.Main

* Note last command, where "Main" class is kept in some package. While executing giving full name of class (pkgname + classname) is expected.

+ CLASSPATH:

- CLASSPATH is an environment variable containing list of directories separated by ; (win) or : (UNIX).

- Java tools (like java compiler, java app launcher, java debugger, etc) and JVM search java classes (.class) and packages into all directories mentioned in the CLASSPATH variable.

- To see current CLASSPATH variable:

[WIN] set CLASSPATH

[LINUX] echo $CLASSPATH

- To add a dir path in in CLASSPATH variable:

[WIN] set CLASSPATH=new_dir_path;%CLASSPATH%

[LINUX] export CLASSPATH=new_dir_path:$CLASSPATH

- To unset/nullify CLASSPATH variable (usually done before staring new assign):

[WIN] set CLASSPATH=

[LINUX] export CLASSPATH=

- If CLASSPATH is not defined, then by default java tools will search classes (.class) and packages in current directory.

Day 04 (3 hours)

+ Members access modifiers:

- public:
  - in class
  - in package
  - in sub-class
  - out-side class
- protected:
  - in class
  - in package
  - in sub-class
- default:
  - in class
  - in package
- private:
  - in class

+ "instanceof" operator

- boolean flag = ref instanceof ClassName;
- If object addressed by given reference is of given class or its sub-class, then instanceof operator results in "true"; otherwise it results "false".
- e.g. Person p1 = new Manager();

```
flag = p1 instanceof Manager; // true
flag = p1 instanceof Employee; // true
flag = p1 instanceof Person; // true
flag = p1 instanceof SalesManager; // false
```

```
flag = p1 instanceof Student; // false
```

+ java.lang.Object class:

- All java classes are directly or indirectly inherited from "Object" class.
- "Object" class is root of java class hierarchy.
- It provides following methods which can reused/overridden into the derived classes:

1. public String toString();
   - This method returns String representation of the object.
   - Programmer should override this method to return appropriate string reprsentation (as per his/her requirement).
   - If not overridden, Object class toString() will be invoked. This method returns a string containing "classname@hashcode".
   - class Person {

```
private String name, address;
private int age;
// ...
public String toString() {
    return String.format("name:%s, addr:%s, age:%d", name, address, age);
}
}
```

   - class Main {

```
public static void main(String[] args) {
    Person p1 = new Person("A","b",3);
    System.out.println( p1.toString() );
    System.out.printf("%s\n", p1);
}
}
```

2. public boolean equals(Object obj);
   - equals() method compares current object to the given object and return true if they are equivalent. Otherwise it returns false.
   - Programmer should override this method to compare objects of class type.
   - If not overridden, Object class equals() will be called. It returns true if references are pointing to the same object; otherwise it returns false.
   - class Employee {

```
private int empid;
private double salary;
// ...
```

```
        public boolean equals(Object obj) {
                if(obj==null)
                        return false;
                if(this==obj)
                        return true;
                if(obj instanceof Employee) {
                        Employee other = (Employee)obj;
                        if(this.empid==other.empid)
                                return true;
                }
                return false;
        }
}
- class Main {
        public static void main(String[] args) {
                Employee e1 = new Employee(...);
                Employee e2 = new Employee(...);
                if(e1.equals(e2))
                        System.out.println("same");
                else
                        System.out.println("not same");
        }
}
```

- equals() method characteristics:
- reflexive: obj.equals(obj) must return true.
- symmetric: obj1.equals(obj2) return true, then obj2.equals(obj1) must return true.
- trasitive: obj1.equals(obj2) return true and obj2.equals(obj3) return true, then obj1.equals(obj3) must return true.
- consistent: obj1.equals(obj2) should produce same results until any one of the object is modified.
- Comparision with null, must return false.
- equals() method should not throw any exception (even if given object is of incompatible type).

3. protected void finalize();
4. public final int hashCode();
5. protected Object clone();
6. public void wait();
7. public void notify();
8. public void notifyAll();

Day 05 (5 hours)

==========

+ Assignment Discussion

```
  + Employee class
public double static calculateTotalSalary(Employee emps[]) {
      double total = 0.0;
      for(int i=0; i<emps.length; i++) {
            total += emps[i].getTotalSalary();
      }
      return total;
}
  + Main class
      Employee[] arr = new Employee[5];
      for(i=0; i<5; i++) {
            sysout("1.labor, 2. mgr, 3. vp, 4. salesmgr, 5. mktmgr : ");
            choice = sc.nextInt();
            switch(choice) {
                  case 1:
                        arr[i] = new Labor();
                        break;
                  // ...
            }
            arr[i].accept();
      }

      double expense = Employee.calcTotalSalary();

+ Enum in java:
      - enum in C/C++:
            - Enum is used to increase readability of the program.
            - Enum constants are internally integer values.
            - enum color {
                  red, green, blue
            };
            - enum color c = red;
            - switch(c) {
```

```
            case red:
                break;
            case green:
                break;
            case blue:
                break;
        }
```

- In above example, enum constants red, green and blue are equivalent to ints 0, 1 and 2 respectively.

- In java, enum is added from version j2se 5.0.
- Like C/C++, enum is user defined type.
- Enum can be declared as:

```
enum Operation {
        Add, Subtract, Multiply, Divide
}
```

- By default all user defined enum types are inherited from java.lang.Enum class.
- All enum constants are internally public static final (constant) members of defined enum type.
- List of all enum objects can be accessed using static method in enum type:

  public static Operation[] values();
- Each enum object can invoke two non-static methods:
  - public int ordinal();
    - return position of enum const in enum declaration
  - public String name();
    - return String representation/name of enum constant.
- Stringized name of enum constant can be converted into enum const object using static method in enum type:

  public static Operation valueof(String str);

+ abstract class and method:

- If you donot want to create object of a class, that class can be declared as "abstract".
- public abstract class ClassName {

```
    // ...
}
```

- If you donot want to implement certain method in the class, then method can be declared as "abstract".
- As abstract method doesn't have implementation, the class is incomplete. So we cannot create object of such class.
- Thus, if a class contains atleast one abstract method, class must be declared as "abstract".

- public abstract class Employee ... {

```
    // ...
    public abstract double getTotalSalary();
}
```

- If method is abstract in the class, sub-class is forced to override that method. If not overridden, method will remain abstract in sub-class and sub-class should also be declared as abstract.
- The abstract class can have fields, constructors and other methods too. All these members can be reused from the sub-class.

+ final keyword:

- final local variables:
  - Local variables (within method) can be declared as final.
  - The final variables can be initialized only once (may or may not be at the point of declaration).
  - The variable cannot be modified once initialized.
  - void fun1() {

```
        final int num = 100; // primitive type
        // ...
    }
```

  - void fun2() {

```
        final Person p1 = new Person();
        // p1.setName("New Name");    // allowed
        // p1 = new Person(); // error
    }
```

- In above example, p1 is constant reference and cannot be modified (i.e. cannot point to another object).
- final arguments:
  - void fun(final Person p1) {

```
        // ...
    }
```

  - p1 is constant reference and cannot be modified.
- final fields:
  - Non-static final fields can be initialized at the point of declaration in the class or within constructor.
  - Static final fields can be initialized at the point of declaration in the class or static block.
  - However final fields cannot be modified after initialization.
  - class Circle {

```
        public static final double PI = 3.14;
        // ...
    }
```

```
- class Employee {
        private final int empid;
        // ...
        public Employee(int empid, ...) {
                this.empid = empid;
                // ...
        }
}
```

- **final methods:**
  - If method is declared as final, it cannot be overridden in sub-class. Otherwise compiler raise error.
  - e.g. public final Class getClass();
    is a final method in java.lang.Object and hence it cannot be overridden.
- **final class:**
  - The final class cannot be inherited. If tried so, compiler raise error.
  - e.g. String class is final class.
  - e.g. public final class VicePresident {
    // ...
    }
  - All methods in final class are treated as final methods.

+ **interface:**
  - Interface contains only method declarations (no implementations, no fields, no constructors, etc).
  - All methods in interface are public and abstract, which force all inherited classes to implement them.
  - Thus the caller can invoke any of the interface method on the implementing class safely (as method will be definately available there).

  - In java, interface is declared using "interface" keyword.
  - All methods are public and abstract. Declaration of "public" and "abstract" keywords is optional.
  - Interface fields (if present) are public static and final. (Declaring public static final is optional).
  - While inheriting class from interface, "implements" keyword is used.
  - A class can be inherited from one or more interfaces.
  - **Syntax:**
    - If A, B, C are classes and I1, I2, I3 are interfaces:

```
class A extends B                    // valid
class A extends B, C                 // invalid
class A implements I1                // valid
class A implements I1,I2             // valid
class A implements I1 extends B      // invalid
```

class A extends B implements I1 // valid
class A extends B implements I1,I2 // valid
interface I3 implements I1 // invalid
interface I3 extends I1 // valid
interface I3 extends I1, I2 // valid
interface I3 extends C // ????

- Interfaces are used to give standard/specification.

```
interface Shape {
	public static final double PI = 3.14;
	// double PI = 3.14;
	public abstract double calcArea();
	double calcPeri();
}
- class Circle implements Shape {
	private double radius;
	// constructor
	public double calcArea() {
		return Shape.PI * radius * radius;
	}
	public double calcPeri() {
		return 2 * Shape.PI * radius;
	}
}
- class Main {
	public static void main(String[] args) {
		Shape ob = new Circle(7);
		System.out.printf("area : %f\n", ob.calcArea());
	}
}
```

Day 6
=========

```
+ Q. equals() method use??
	public static int search(Object[] arr, Object p) {
		for (int i = 0; i < arr.length; i++) {
			if(p.equals(arr[i]))
```

```
                return i;
        }
        return -1;
    }
    public static void main(String[] args) {
        Student[] arr = new Student[5];
        arr[0] = new Student( ... );
        arr[1] = new Student( ... );
        arr[2] = new Student( ... );
        arr[3] = new Student( ... );
        arr[4] = new Student( ... );
        System.out.println("Enter roll to search : ");
        int roll = sc.nextInt();
        Student s = new Student();
        s.setRoll(roll);
        int index = search(arr, s);
        if(index==-1)
            System.out.println("Record Not Found");
        else
            System.out.println("Record Found");
    }
```

+ swapping two objects:

```
    public static void swap(Distance d1, Distance d2) {
        int temp;
        temp = d1.feet;
        d1.feet = d2.feet;
        d2.feet = temp;
        temp = d1.inches;
        d1.inches = d2.inches;
        d2.inches = temp;
    }
    public static void main(String[] args) {
        Distance d1 = new Distance(11, 6);
        Distance d2 = new Distance(8, 10);
        System.out.printf("d1:%s, d2:%s\n", d1, d2);
        swap(d1, d2);
```

```
        System.out.printf("d1:%s, d2:%s\n", d1, d2);
    }
```

+ **java.util.Comparable interface:**

- This is standard way of comparing two java objects.
- To sort array of objects using
  Arrays.sort() method
  The object's class must be inherited from Comparable interface.
- This interface has single method:
  int compareTo(Object obj);
  - If "this" object is equal to "obj", return 0.
  - If "this" object > "obj", return +1, else return -1.
- class Student implements Comparable {

```
    // ...
    public int compareTo(Object obj) {
        Student other = (Student) obj;
        if(this.roll == other.roll)
            return 0;
        if(this.roll > other.roll)
            return +1;
        else
            return -1;
    }
}
```

- The sort() method can be called as follows:

```
Student[] arr = new Student[5];
arr[0] = new Student(...);
// ...
arr[4] = new Student(...);

Arrays.sort(arr);
```

- Arrays.sort() method sorts the array of objects. Whenever comparision of two objects is required during sorting process, it internally calls compareTo() method on the object.
- If class is not inherited from Comparable interface, Arrays.sort() throws ClassCastException.

+ user-defined sorting method:

```
public static void sort(Object[] arr) {
    for(int i=0; i<arr.length; i++) {
        for(int j=i+1; j<arr.length; j++) {
            Comparable arri = (Comparable)arr[i];
            if(arri.compareTo(arr[j]) > 0) {
                Object temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

+ Marker interfaces:

- In java, interfaces may not have any method declarations (i.e. empty interfaces). Such interfaces are called as "marker interfaces".
- They are used to indicate that the class is associated with certain functionality.
- e.g. Serializable is a marker interface. If class is inherited from "java.io.Seriablizable" interface, then JVM ca convert object details into sequence of bytes (using ObjectOutputStream class's write object method). If class is not inherited from Serializable, JVM will throw exception.
- e.g. Cloneable is a marker interface. If class is inherited from "Cloneable", then Object class's clone method can create copy of the object of that class. Otherwise it throws CloneNotSupportedException.
    - example: class Date, class Person, ...

+ Exception Handling:

- Runtime problems are known as exceptions. Exception handling is flexible way of handling runtime problems in the execution of the program.
- When a function cannot handle a problem detected at runtime, it will "throw" it in form of object of class.
- The calling function should use "try-catch" block for handling such runtime problems.
- If no runtime problem occur, all statements in try block will be executed.
- If runtime problem occurs, all statements thereafter in try block are skipped and catch block is executed.

- In java, runtime problems can be thrown in form of objects of the classes inherited from "java.lang.Throwable" class.
- There two main types of Throwables:
- **Error**
    - Represents non-recoverable JVM-internal problems (usually).

- All error classes are inherited from java.lang.Error.
- e.g. java.lang.OutOfMemoryError, etc.

- **Exception**
  - Represents runtime problems that can be handled
  - There are two types of exceptions:
  - **unchecked exceptions**
    - Mostly occur due to programmer/end user's mistakes.
    - All such exception classes are inherited from java.lang.RuntimeException.
    - Compiler do not force to handle these exception.
    - e.g. NullPointerException, ArrayIndexOutOfBoundsException, etc.
  - **checked exceptions**
    - Represents runtime problems raise within JVM or outside the JVM.
    - Compiler force to handle these exceptions in one of the following ways:
    - Calling function should have try-catch block.
    - Calling function has throws clause specifying exception class.
    - All class inherited from java.lang.Exception (except RuntimeException and its sub-classes) represents checked exceptions.
    - e.g. IOException, SQLException, CloneNotSupportedException, etc.

- **Java Exception Hierarchy:**

```
- Throwable
    |- Error
        |- OutOfMemoryException
        |- etc.
    |- Exception
        |- IOException
        |- SQLException
        |- CloneNotSupportedException
        |- etc.
        |- RuntimeException
            |- NullPointerException
            |- ArrayIndexOutOfBoundsException
            |- ClassCastException
            |- etc.
```

- If exception is not handled by the calling function, then it will propogated to its calling function. However if exception is not handled in the entire chain of functions. finally it is given to main() function.
- If main() function doesn't handle the exception, it will be caught by the JVM; however program will be terminated.

- One try-block can have multiple catch blocks. However exception sub-class catch block should appear first and then exception super-class catch block. Otherwise compiler will raise error.

- e.g.

```
try {
        // ...
} catch(ArrayIndexOutOfBoundsException e) {
        // ...
} catch(RuntimeException e) {
        // ...
} catch(Exception e) {
        // ...
} catch(Throwable e) {          // generic catch
        // ...
}
```

- **In java, finally block can be written after catch block(s). It is optional block.**

```
- try {
        // ...
} catch(Exception e) {
        // ...
} finally {
        // ...
}
```

- The finally block will always execute irrespective of exception occur or not.
- Even if control is returning/jumping from try or catch block (as a result of return, break or continue like statements), finally block is still executed.
- The only case in which finally block will not be executed in execution System.exit() within try or catch block.

```
- example:
class Main {
        public static int divide(int num, int den) {
                if(den==0)
                        throw new RuntimeException("Den cant be 0.");
                return num / den;
        }
        public static void main(String[] args) {
                int num, den, res;
```

Prasanna wadekar


```
            Scanner sc = new Scanner(System.in);
            try {
                num = sc.nextInt();
                den = sc.nextInt();
                res = divide(num, den);
                System.out.printf("Result : %d\n", res);
            } catch(RuntimeException e) {
                System.out.println("Error occured!");
            } finally {
                System.out.println("Always executed!");
            }
        }
    }
```

- java.lang.Throwable:
  - public Throwable(String message);
    - Throwable class object encapsulates a string message, which can be initialized via parameterized constructor. Such constructor is present for all its derived exception types.
  - public String getMessage();
    - Returns a message encapsulated by exception object.
  - public void printStackTrace();
    - Displays info about exception - mainly exception class type and location in the code from where it is generated (along with stack trace).
  - public String toString();
    - returns exception details in form of string which contains exception type, message.
- User-defined exception class:
  - example:

```
class MyException extends RuntimeException {
    private String field;
    private int value;
    public MyException(String msg, String field, int value) {
        super(msg);
        this.field = field;
        this.value = value;
    }
    // getters
}
class Date {
```

```
        // ...
        public void setYear(int year) {
                if(year < 0)
                        throw MyException("invalid year", "year", year);
                this.year = year;
        }
}
```

- throws clause : Exception specification list
  - example:

```
class Temp {
        public void fun() throws IOException, ServletException {
                // ...
        }
}
```

- If a method is not handling generated checked excepion (using try-catch block), then it must write throws clause in method prototype to indicate exception types it may throw.
- if class B extends A:

```
A -> void fun() throws E1, E2, E3;
```

then

```
B -> void fun() throws E1, E2, E3; // OK
```

OR

```
B -> void fun() throws E1, E2;          // OK
```

BUT

```
B -> void fun() throws E1, E4;          // Error
```

In other words, an overridden sub-class method can throw same or fewer exceptions than corresponding super-class method; but it cannot throw any additional exception.

- if class D extends C:

```
C -> void fun() throws IOException;
```

then

```
D -> void fun() throws IOException;     // OK
```

OR

```
D -> void fun() throws EOFException;// OK
```

BUT

```
D -> void fun() throws Exception;       // Error
```

In other words, an overridden method can throw exception of the same type or exception sub-class typ corresponding to super-class method; however it cannot throw super-class of exception type.

Day 07

==========

```
- multi-catch block: (j2se 7)
    - example:
        try {
            // ...
            fun();
            // ...
        } catch(EOFException e) {
            // igonore : do nothing
        } catch(IOException | SQLException e) {
            System.out.println("Error Occured!");
            System.Exit(1);
        } catch(Exception e) {
            e.printStackTrace();
        }
```

- When we want to implement same logic to handle more than one exception, we can use multi-catch block as shown in above example.

+ Wrapper classes:

- In java, primitive types are not classes i.e. primitive type variables are not objects.
- However sometimes, there is need to treat prmitive type values as java objects. E.g. java collections can work only with objects (not with primitive types).
- Example: till j2se 1.4

```
ArrayList obj = new ArrayList();
// It has add method to add object of any type
// void add(Object ob);
obj.add(11);      // error: cannot add primitive type
obj.add(new Integer(11)); // no error: Integer obj
```

- Thus "Integer" is wrapper class provided by java to treat "int" values as "objects".
- Java provides wrapper classes for all primitive types as follows:
    - byte -> Byte
    - short -> Short
    - int -> Integer
    - long -> Long
    - float -> Float
    - double -> Double

- char -> Character
- boolean -> Boolean

- Wrapper class objects encapsulate primitive type values. However additionally they provide lot of helper methods (static / non-static). Also they define lot of helpful constants.

+ Autoboxing:
- example;

```
int num1 = 100;
Integer num2 = num1;
```

- This is valid syntax from J2SE 5.0 and onwards.
- Here primitive type value is automatically converted to corresponding wrapper class object type. This feature is known as "autoboxing".
- example:

```
int num3 = num2;
```

- As shown in above example, even wrapper class object can be converted back to the corresponding primitive type value directly.
- However conversion from primitive type to wrapper object is not efficient (as internally memory is allocated, as well as it takes time). For better performace autoboxing should avoided whereever possible.
- example:

```
Integer i1=10, i2=20, i3;
i3 = i1 + i2;
```

- example:

```
ArrayList obj = new ArrayList();
obj.add(11);      // valid from j2se5.0 - but slow
```

+ String class:

- String encapsulate set of characters and length. It provides numerous method for manipulating String data.
- String is immutable object. Any method modifying string contents will not change the object's contents; rather it creates a new String object with modified contents.
- example:

```
public static void main(String[] args) {
    String s1 = "SunBeam", s2;
    System.out.printf("s1 : %s\n", s1);
    s2 = s1.concat("Infotech");
    System.out.printf("s1 : %s\n", s1);
    System.out.printf("s2 : %s\n", s2);
}
```

- **String Comparision:**

- Two object's equality can be checked using equals() method.
- They can also be compared using Comparable interface compareTo() method.
- Object references are compared using == operator. If this operator results true, both reference are pointing to same object.

- String class overrides equals() method and implements Comparable interface.
- String s1 = "sunbeam", s2="infotech";
    - Internally two String objects are created with different values.
    - s1==s2 => false
    - s1.equals(s2) => false
- String s1 = "sunbeam", s2=new String("sunbeam");
    - Use of new operator force to create a new object with similar contents so:
    - s1==s2 => false
    - s1.equals(s2) => true
- String s1 = "sunbeam", s2="sunbeam";
    - To minimize memory requirement, java compiler add a single entry into "string table" maintained in .class file. Obviously at runtime single string object is created and both references point to the same.
    - s1==s2 => true
    - s1.equals(s2) => true
- String s1 = "sunbeam", s2="sun"+"beam";
    - Java compiler itself add "sun" and "beam" to create a new string "sunbeam" and found that the entry is already present in string table. So again single object is created and both references point to it.
    - s1==s2 => true
    - s1.equals(s2) => true

- Java does not support operator overloading; however +, += operators are built-in overloaded for String class.
- String methods:
    - String(String s);
    - String(char []arr);
    - int length();
    - char charAt(int index);
    - boolean equals(Object str);
    - boolean equalsIgnoreCase(String str);
    - int compareTo(String str);
    - static String format(String fmt, ...);
    - int indexOf(char ch);
    - int indexOf(String substr);

- String substring(int start, int end);
- String[] split(String regex);
- String concat(String s);
- String replace(String find, String replace);
- String toUpperCase();
- String toLowerCase();

+ StringBuffer Class:

- Represents mutable string object i.e. any method modifing contents of object will modify contents of the same object.
- example:

```
public static void main(String[] args) {
    StringBuffer s1 = new StringBuffer("sunbeam");
    System.out.printf("s1 : %s\n", s1.toString());
    s1.append("infotech");
    System.out.printf("s1 : %s\n", s1.toString());
}
```

- StringBuffer class encapsulates a string buffer which can dynamically grow as number of characters in the object are increased (appended).
- The number of characters present can be found using length() method, while maximum number of characters that can be stored in internal buffer can be found using capacity().
- **Stringbuffer Methods:**
  - StringBuffer(String s);
  - int length();
  - int capacity();
  - char charAt(int index);
  - StringBuffer append(String s);
  - StringBuffer reverse();
  - etc.
- All methods in StringBuffer class are synchronized, hence it can be safely manipulated in multi-threaded program. However due to synchronization, efficiency reduces (slower).

+ Stringbuilder Class:

- Functionalitywise same as StringBuffer.
- Added into J2SE 5.0.
- Its methods are not synchronized. So used in single threaded program to improve the performance.

+ Stringtokenizer Class:

- Use to split string data into multiple parts. Nowdays split() method is preferred over StringTokenizer.

+ java.util.Calendar class:

- Represents a calendar date and time.
- To create the object:
  - Calendar cal = Calendar.getInstance();
  - By default object will contain current date and time values.
  - Calendar is abstract class. On desktop (J2SE), it internally creates object of java.util.GregorianCalendar class.
- To set calendar fields:
  - cal.set(year, month, day);
  - cal.set(year, month, day, hour, min, sec);
  - Imp note: months are counted from 0 to 11.
- To get value of individual field:
  - int dd = cal.get(Calendar.DAY_OF_MONTH);
  - int mm = cal.get(Calendar.MONTH) + 1;
  - int yy = cal.get(Calendar.YEAR);

+ Java.Util.Date Class:

- Was used to represent a calendar day & time. Nowdays most of the functions from this class is deprecated.
- Currently this class is used to represent time duration in milliseconds from epoch time i.e. 1-JAN-1900 00:00:00.
- example:

```
Date d2 = new Date();
System.out.println(d2.toString());
```

+ Nested Classes:

1. static member class:
   - A class can be declared within another class with "static" keyword. In this case, the inner class is treated as a static member of the outer class.
   - Inner class object is not dependent on outer class object.
   - Inner class can access static members of outer class directly; however it cannot access non-static members directly.
   - If name of static member is same in inner and outer class, then outer class member can be accessed in inner class using syntax "Outer.member".
   - static member class can have public, private, protected or default access modifier.
   - If static member is visible outside the Outer class (due to its access modifier), then it can be accessed using syntax "Outer.Inner" outside the class.

- Outer.Inner obj = new Outer.Inner();
- example:

```
class Outer {
        static class Inner {
                // ...
        }
        // ...
}
```

2. non-static member class:

- Non-static inner class object can be created, if and only if it is associated with some outer class object.
- Non-static member class can access static as well as non-static members of the outer class directly.
- Non-static member class have a hidden reference to the associated outer class object i.e. "Outer.this".
- If name of non-static member in outer class is same as that of inner class, then it can accessed in the inner class using syntax "Outer.this.member".
- Object of inner class can be created outside the Outer class, using one of the following way:
  - Outer obj = new Outer();
    Outer.Inner ref1 = obj.new Inner();
  - Outer.Inner ref2 = new Outer().new Inner();
- Inner class object can be created in the outer class non-static method directly:
  - class Outer's non-static method:

```
void nsMethod() {
        Inner obj = new Inner();
        //Inner obj = this.new Inner();
}
```

- example:

```
class Outer {
        class Inner {
                // ...
        }
        // ...
}
```

+ example:

```
class List implements Iterable {
        private static class Node {
                Object data;
                Node next;
                public Node() {
```

```
            this(null);
        }
        public Node(Object val) {
            this.data = val;
            this.next = null;
        }
    }

    private Node head;
    public List() {
        this.head = null;
    }
    public void add(Object val) {
        Node newnode = new Node(val);
        if(head==null)
            head = newnode;
        else {
            Node trav = head;
            while(trav.next!=null)
                trav = trav.next;
            trav.next = newnode;
        }
    }

    private class Iter implements Iterator {
        private Node nodeObj;
        public Iter() {
            this.nodeObj = head;
        }
        public boolean hasNext() {
            return nodeObj!=null ? true : false;
        }
        public Object next() {
            Object val = nodeObj.data;
            nodeObj = nodeObj.next;
            return val;
        }
```

```
            public void remove() {
            }
        }

        public Iterator iterator() {
            return this.new Iter();
        }
    }
    public class Main {
        public static void main(String[] args) {
            List obj = new List();
            obj.add("A");
            // ...
            List.Iter itr = obj.iterator();
            while(itr.hasNext()) {
                String val=(String)itr.next();
                System.out.println(val);
            }
            for(Object val : obj)
                System.out.println(val);
        }
    }
```

Day 8

3. local class:

- In java, classes can be defined within the method. Such classes are treated like local variables and are called as "local classes".
- Their scope is limited to the method in which they are implemented. In other words, the class cannot be accessed outside the enclosing method.
- If class is implemented within static method, it behaves as static member class (i.e. can access static members of outer class directly).
- If class is implemented within non-static method, it behaves as non-static member class (i.e. can access static and non-static members of outer class directly).
- In both cases, local classes can access final local variables of the enclosing method.
- example:

```
class Main {
```

```
        public static void main(String[] args) {
            Student[] arr = new Student[5];
            arr[0] = new Student(...);
            // ...

            class StudNameComparator implements Comparator {
                public int compare(Object o1, Object o2){
                    Student s1=(Student)o1;
                    Student s2=(Student)o2;
                    int diff = s1.getName().compare(s2.getName());
                    return diff;
                }
            }
            StudNameComparator cmp = new StudNameComparator();
            Arrays.sort(arr, cmp);

            for(Student s:arr)
                System.out.println(s);
        }
    }
```

- We can create any number of objects of local class within enclosing method.

4. Anonymous Inner Class:

- example:

```
ClassName obj = new ClassName() {
    // ...
};
```

- When you want to create single object of a class inherited from some other class/interface and need to invoke its overridden methods, anonymous class would be the best choice.
- In above syntax example, internally a new class is created inherited from "ClassName" and then its object is created, which is accessed by "obj" reference.
- Obviously with "obj" reference we can access only overridden methods of that inherited anonymous class.
- Usually such classes are written in some method and hence their behaviour is same as "local classes".
- example:

```
Arrays.sort(arr, new Comparator() {
    public int compare(Object o1, Object o2) {
        Student s1 = (Student)o1;
        Student s2 = (Student)o2;
```

```
                int diff = s1.getName().compareTo(s2.getName());
                return diff;
            }
        });
```

+ **Generics in Java:**
  - When same logic need to be implemented for different data types, generic (template) programming can be used in java.
  - Example: All data structures (e.g. Stack, Queue, List, Tree, etc) logic remains same irrespective of which type of objects are used in that data structure.
  - **Limitations of generics in Java:**
    1. Cannot be used with primitive types.
    2. Cannot create array of generic type (T).
    3. Type of generic type cannot be properly detected at runtime using reflection. Also cannot work well with "instanceof" operator.
  - Efficiency of generic types is same as efficiency of corresponding "Object" based types in Java.
  - **Advantages:**
    1. The same generic class can be reused for different class types.
    2. Generic classes are type-safe. If different type of elements is added in the generic collection, compile time error is raised.
  - Generics were added in Java 5.0 version. Till Java 1.4 all collection classes were used as "Object" collections. However 5.0 onwards, all collection classes refined to take generic parameters.

    Ex1.4: ArrayList list = new ArrayList();
    Ex5.0: ArrayList<Person> list = new ArrayList<Person>();
    Ex7 : ArrayList<Person> list = new ArrayList<>();

  - example:

```
class Stack<T> {
    private T[] arr;
    private int top;
    public Stack(Class type, int size) {
        arr = (T[])Array.newInstance(type, size);
        top = -1;
    }
    public void push(T ele) {
        arr[++top] = ele;
    }
    public void pop() {
```

```
                top--;
            }
            public T peek() {
                return arr[top];
            }
            public boolean empty() {
                return top==-1;
            }
            public boolean full() {
                return top==arr.length-1;
            }
        }
        public class Main {
            public static void main(String[] args) {
                Stack<Integer> s1 = new Stack<>();
                s1.push(11);
                s1.push(22);

                int ele;
                ele = s1.peek();
                s1.pop();
                System.out.printf("popped : %d\n", ele);
            }
        }
```

- **Generic Interfaces:**
  - predefined java.util.Comparable<T> interface:

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

  - implementing Comparable<T> in Student class:

```
class Student implements Comparable<Student> {
    // ...
    public int compareTo(Student other) {
        if(this.roll==other.roll)
            return 0;
        return this.roll > other.roll ? +1 : -1;
    }
```

```
}
```

- **Generic Methods:**

```
- By default all methods in generic class are generic methods.
- However it is possible to implement generic method in non-generic
  class.
- example:
class Util {
    public static void swap<T>(T o1, T o2) {
        T t = o1;
        o1 = o2;
        o2 = t;
        System.out.printf("o1:%s, o2:%s\n", o1, o2);
    }
}
class Main {
    public static void main(String[] args) {
        String s1="sunbeam", s2="infotech";
        Util.swap(s1,s2);
        // ...
    }
}
```

+ Java Collection Framework:

- Collection Framework = Interfaces + Implementations + Algorithms

+ Java Collection Hierarchy:

- All collection interfaces and classes are kept under java.util package.

```
+ Collection<E>
    |- Set<E>
        |- HashSet<E>
        |- LinkedHashSet<E>
        |- TreeSet<E>
    |- List<E>
        |- ArrayList<E>
        |- LinkedList<E>
        |- Vector<E> *
            |- Stack<E>
```

|- Queue<E>

|- PriorityQueue<E>

+ Map<K,V>

|- HashMap<K,V>

|- LinkedHashMap<K,V>

|- TreeMap<K,V>

|- Hashtable<K,V> *

* Before J2SE 1.2 only Vector and Hashtable were supported. When collection framework is introduces with JAVA2, these classes are modified to be compatible with collection framework interfaces.

+ Java Collection Interfaces:

1. Collection<E>
   - Root of collection hierarchy.
   - Most of the classes inherited from Collection<> interface.
2. Set<E>
   - Is a collection which does not allow duplicate values.
3. List<E>
   - Is a collection which allows duplicate values, bidirectional traversing and random access.
4. Queue<E>
   - Is a collection which provide data structure queue functionality.
5. Map<K,V>

- Not inherited from Collection interface.
- Stores data in key, value pair so that for a given key, value can be searched in fastest possible time.

+ interface Collection<E>:

- extends Iterable<E>
  - Iterator<E> iterator();
  - All classes inherited from Collection interface, internally implement Iterator.
  - They all can be traversed using for-each loop.

- boolean add(E ele);
- boolean contains(E ele);
- boolean remove(E ele);
- boolean isEmpty();
- int size();

- boolean addAll(Collection<E> c);

- boolean containsAll(Collection<E> c);
- boolean removeAll(Collection<E> c);
- boolean retainAll(Collection<E> c);
- void clear();

+ interface Set<E>

- HashSet<E>
  - Elements are stored in any order (based on hashcode).
  - Faster/Efficient search.
  - Equality is based on hashCode() and equals() method.
- LinkedHashSet<E>
  - Elements are stored in the order of insertion.
  - Equality is based on hashCode() and equals() method.
- TreeSet<E>
  - Elements are stored in the sorted order.
  - Equality and comparision is based on compareTo() method of Comparable interface.
  - In other words, objects to be added into TreeSet<> should implement Comparable interface; otherwise ClassCastException will be thrown.
  - Note that TreeSet<> does not depend on equals() and hashCode() method of the containing objects.
  - As a standard practice, if equals() and compareTo() both are implemented, then both should perform comparision on the same fields.

Day 9

==========

+ interface List<E>

- Since it is inherited from Collection<E> interface, it has all methods discussed in Collection<E>.

- List<> provides additional searching mechanisms:
  - int indexOf(Object o);
    - Compare given object with all objects in the list using equals() method and if found, return "index" of that element. If not found return -1.
  - As list can have duplicate elements, this method returns first occurance of the element
    - example:

```
roll = sc.nextInt();
s = new Student();
s.setRoll(roll);
```

```
index = list.indexOf(s);
if(index!=-1)
        System.out.println("Found");
else
        System.out.println("Not Found");
```

- int lastIndexOf(Object o);
- Compare given object with all objects in the list in the reverse order using equals() method. If found return index of object, else return -1.
- As list can have duplicate elements, this method returns last occurance of the element.

+ Random access:

- Element at any index can be accessed/replaced directly.
- E get(int index);
  - Returns object at given index. Throws exception for invalid index.
  - example:

```
index = sc.nextInt();
s = list.get(index);
System.out.println(s);
```

- E set(int index, E newEle);
  - Set a new object at given index (replacing old object).
  - example:

```
index = sc.nextInt();
s = new Student();
s.accept();
list.set(index, s);
```

- E remove(int index);
- deletes element at given index and all elements after that index will be shifted to index-1.

+ Bi-directional Traversing:

- List<> internally implements iterator for traversing. This iterator is inherited from ListIterator<> interface.
  - ListIterator<> interface is inherited from Iterator<>. It has additional methods:
    - boolean hasPrevious();
    - E previous();
    - void set(E ele);
  - Object of ListIterator<> can be received using listIterator() method of List<> interface.
    - ListIterator<> listIterator();
    - ListIterator<> listIterator(int index);

- **Forward traversing:**

```
ListIterator itr = list.listIterator();
while(itr.hasNext()) {
        Student s = itr.next();
        System.out.println(s);
}
```

- **Reverse traversing:**

```
ListIterator itr = list.listIterator(list.size());
while(itr.hasPrevious()) {
        Student s = itr.previous();
        System.out.println(s);
}
```

- By default, iterator points to start of the list, so reverse traversal is not possible.
- So in case of reverse traversing, list iterator should begin from end of list (as shown in above example).

+ **Algorithms which can work on List<> iterface:**
  - All methods given here are static methods of java.util.Collections class.
  - Sorting List<>:
    - void sort(List<T> list);
      - The type T must be inherited from Comparable<>.
    - void sort(List<T> list, Comparator<T> c);
  - Reversing List:
    - void reverse(List<T> list);
  - Randomizing List:
    - void shuffle(List<T> list);
  - Binary Search:
    - List should be initially sorted.
    - The type T must be inherited from Comparable<>.
    - void binarySearch(List<T> list, T obj);

+ List<> Implementations:
  - **ArrayList<> class:**
    - Internally implemented as doubly linked list of arrays. Each node contains multiple elements.
      - It gives more efficient random access.
      - Insertion and deletion (at middle of list) is slower operation).
- **LinkedList<> class:**
- Internally implemented as doubly linked list with head and tail pointer.

- Insertion and deletion at ends of list are very fast. Also insertion and deletion at mid of list if efficient.
- However random access is slower.
- **Vector<> class:**
    - Vector<> is implemented as dynamic array (contiguous memory). The array grows or shrink runtime as per requirement.
    - However Vector<> is synchronized class (all methods are synchronized).
    - It is mainly preferred in multi-threaded program. However sync reduce the performance in single threaded program.
        - **Stack<> class:**
            - Inherited from Vector<>.
            - Provides last-in-first-out behaviour using methods: push(), pop(), peek(), ...

+ Queue<> interface:

- Implementation class:
    - PriorityQueue<>.

+ Map<K,V> interface:

- Map does not allow duplicate keys. If try to add duplicate key, new value will replace old value.
- methods:
    - V put(K key, V val);
    - V get(K key);
    - int size();
    - void clear();
    - void remove(K key);
    - boolean containsKey(K key);
    - boolean containsValue(V value);
    - Set<K> keySet();
    - Collection<V> values();
    - Set<Map.Entry> entrySet();
- **Implementation classes:**
    - HashMap<K,V>:
        - Elements are stored in any order of the keys.
        - Faster searching
    - LinkedHashMap<K,V>:
        - Elements are stored in the order of insertion of keys.
    - TreeMap<K,V>:
        - Elements are stored in the sorted order of keys.
        - The Key class must implement java.util.Comparable<K>.

- **Hashtable<K,V>:**
  - Elemenst are stored in any order of keys.
  - However Hashtable<> is synchronized class (all methods are synchronized).
  - It is mainly preferred in multi-threaded program. However sync reduce the performance in single threaded program.

* Note:

Map<Integer,Student> map = ...;

KEY : Integer

VALUE : Student

- KEY class i.e. Integer class must implement Comparable<> interface for TreeMap<> and must implement equals() & hashCode() for HashMap<>, LinkedHashMap<> and HashTable<>.

* Note:

Map<Student,ProgressCard> map = ...;

KEY : Student

VALUE : ProgressCard

- In this case Student class must implement Comparable<> interface for TreeMap<> and must implement equals() & hashCode() for HashMap<>, LinkedHashMap<> and HashTable<>.

+ Time Complexity:

- It is approximate measure of time required to complete any algorithm.
- Many times time complexity is calculated from number of iterations done into the program.
- For example:

  n! = 1 * 2 * 3 * ... * n

  - The program will be having loop repeated n times. So time required is proportional to "n". Thus time complexity is of order of "n" and denoted as O(n).
  - Here "O(n)" is known as Big O notation.
- Few Imp Time complexities:
  - Time required to add or delete element from stack or queue, is constant (independent of number of elements). So time complexity is O(1).
  - Time required to add element at the end of linked list having only head pointer, depends on number of elements (loop is repeated to traverse to last element). So time complexity is O(n).
  - Time complexity of binary search:

    2^i = n

    [where, i is num of iterations and n is number of elements in the sorted array]

    Time complexity O(log n).
  - Time complexity of selection sort:

Number of comparisions = (n-1) + (n-2) + ... + 1

= n (n-1) / 2

So time complexity is O(n^2)

+ Hashing:

- Hashing is used for fast searching.
- The data is stored in form of key-value pair, so that for a given key value can be searched in fastest possible time.
- Ideal time complexity of searching in hash table is O(1).

- Hash table is internally an array, holding data and data is accessed by proper index.
- The values are stored into the array, at a particular index. This index is calculated from the given key using some mathematical function, called as "hash function".
- Simplest example:
  - Storing student info so that it can searched for given roll number.
  - Here array of "n" student objects can be created and information of student with roll number "r" can be stored at index "r-1".
  - This "r-1" is example of simple hash function.
  - h(key) = key - 1 => results in index of hashtable.
- Here student with roll "r" can be immediately found at index "r-1". Time complexity of searching O(1).

- However in the complex examples, multiple keys can result to same index/slot into the hashtable/array. This is called as "collision".
- The collision can handled in one of the following ways:

1. Open Addressing:

- If slot calculated using hash function is already occupied by some object, then another mathematical function is used to find alternative slot for the object. This is called as "re-hashing".
- Simplest re-hashing function would be "index+1" i.e. next slot of table (if empty). This simplest mechanism of open addressing is known as "linear probing".
- Load factor = num of elements / num of slots.
- This mechanism can be used only if Load factor <= 1.0.

2. Chaining:

- Each slot of hash-table holds a list of objects competing for the same slot. This list is known "bucket".
- As number of collisions are increased, the time complexity of searching will be deviated (increased) from O(1).

+ hashCode() method:

- java.util.Object class has hashCode() method, which can be overridden into the derived to return hash code/value of the object.

- This hash code determines the slot of the hashtable, in which object is to be stored.

- If not overridden, Object class hashCode() is called, which return hash value based on memory address of the object.

- **Rules of hashCode() method:**

1. If two objects are equal (by equals() method), then their hash code must be same.
2. If two objects are not equal (by equals() method), then their hash code may be same.

- Hence hash code must be calculated on the same fields, on which equality is compared.

- Statistically it is proven that hash code calculated using prime numbers is uniformly distributed (more unique) and hence most of the hash code calculations are done using prime numbers.

+ Reflection:

- Reflection technique is used to get information about the data type at runtime.

- Three applications/uses of reflection:

1. To get complete info about the type at runtime. It is used to develop intellisence feature, tools like javap.exe.
2. Use the retrieved information about the type to create object of class at runtime and invoke its methods. This is used in most of frameworks like struts, servlets, applets, etc internally.
3. Generate the java classes (byte code) at runtime and execute them. This can be done by using ClassLoader.defineClass() method.

- Information about a type can be received in java.lang.Class object using one of the following way:

1. Class c = obj.getClass();
   - Get the info of the class whose object is referred by "obj".
2. Class c = ClassName.class;
   - Get the info of the class with name "ClassName".
3. Class c = Class.forName("pkg.ClassName");
   - Get the info about the class whose name will be taken at runtime (in form of String).
   - This method loads the class in the JVM and return its information.

- java.lang.Class methods:
  - String getName();
  - boolean isAbstract();
  - boolean isInterface();
  - boolean isEnum();
  - Field[] getFields();
  - Method[] getMethods();
  - Constructor[] getConstructors();
  - Annotation[] getAnnotations();

+ **GUI programming in JAVA:**

- JAVA provides different classes for creating desktop GUI. There are two main ways to create GUI on JAVA:

1. AWT - Abstract Window Toolkit:

- Java UI classes internally depends on the Peer classes, which in turn use OS functions to create UI.
- e.g. java.awt.Button class internally use "ButtonPeer", which in turn calls OS function ("CreateWindow" on Windows OS) to create button UI.
- Due to this AWT UI will be have different look & feel on different platforms (OS).
- Also internal use of peer classes make them heavy components.

2. SWING:

- Java has another set of classes to create platform independent UI. These classes are called as "SWING".
    - The look & feel of UI remains same on all platforms.
    - Swing components internally use MVC architecture i.e. each swing component (ui control class) internally use three objects to split the functionality:
    - Model : keep the data of the component e.g. Strings in the JComboBox.
    - View : keep appearance related info of the component e.g. height, width, back color, fore color, etc.
    - Controller : responsible for calling event handler when event occurs e.g. user click, selection changes in combo box, etc. It also communicate between view and model.

+ **SWING:**

+ Swing component heirarchy:

- Component : represents a UI control on the screen.
- Container : is collection of one or more components.
- Window:
    - Container that can hold multiple components.
    - It has its own window i.e. title bar and border.
- Panel:
    - Container that can hold multiple components.
    - It does not have its own window. It is always present in some other window to group components.
- JFrame:
    - Frame has its own window.
    - So it is stand-alone desktop based application.
    - Frame application has main() function.
- JApplet:
    - Applet does not have its own window.
    - So it always execute within browser window.

- Applet does not have main() function. Its execution begin with init() method.

- Swing components:

- JLabel, JTextField, JTextArea, JButton, JComboBox, JListBox, JCheckBox, etc.

+ Event handling:

- User action is called as an event. e.g. key press, mouse click, etc.
- We can handle these events to add functionality into the GUI.
- The standard way to handle events is to implement appropriate "Listener" interface.
- Example: Button click handler:

* step1: Implement ActionListener interface into some class (in one of the following ways)

- Write a new class inherited from ActionListener interface.
- Implement ActionListener into the user-defined frame class itself.
- Anonymous inner class to implement ActionListener interface.

* step2: Pass the object of class inherited from ActionListener to the addActionListener() method of button object, when that button is created.

* step3: Into this event handling class write appropriate logic within actionPerfromed() method.

- When a button click event occurs, at first it is detected by OS and inform to the JVM. Then JVM pass the event to the button object. Finally button object invokes actionPerformed() method of the listener inherited class object(s) registered with it.

- Event objects:

- Each listener method receive an Event object inherited from "java.awt.Event" class.
- e.g. ActionEvent, WindowEvent, etc.
- This event object contains complete info about the event, which includes source of event (component for which event occured) and additional info.

Day 10

==========

+ Adapter classes:

- Certain listener interfaces have more than one methods. e.g. WindowListener has 7 methods, MouseListener has 6 methods.

- Most of the cases programmer is interested in handling one or two methods from such interface. However being an interface, it is compulsory for the user to implement all the methods.

- To simplify this operation, java provides adapter classes for such interfaces. They are abstract classes having empty/default implementations of all methods in such interface. e.g. WindowAdapter and MouseAdapter.

- So programmer can write his own class inherited from adapter class and override only those methods in which he is interested.

- Typically preferred approach is to create anonymous class inherited from adapter and add its object with target component.

- e.g.

```
// In constructor of the user-defined frame class
this.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
                dispose();
        }
});
```

+ Layout Managers:

- Java has few builtin managers which are used to place components in the container in proper way.
- The layout manager can be set using Container's setLayout() method. If layout manager is not given (i.e. set to null), then programmer must give appropriate x,y coordinates and width,height using setBounds() method of the component.

```
this.setLayout(null);
```

- There are few imp layout managers as follows:

1. FlowLayout:

- Add components in horizontal direction one after another, just like web page.
- this.setLayout(new FlowLayout());
-
```
button = new Button("Caption");
this.add(button);
```
- This is default layout of Panel and Applet class.

2. GridLayout:

- Add components in the grid defined by the programmer. Components will be places in appropriate cell (row & col) one after another.
- this.setLayout(new GridLayout(2,3));
-
```
button = new Button("Caption");
this.add(button);
```

3. BorderLayout:

- Add components in certain direction or center of the container as shown in diag.
- this.setLayout(new BorderLayout());

```
button = new Button("Caption");
this.add(button, BorderLayout.CENTER);
```

- This is default layout of Window and Frame class.

4. GridBagLayout:

- Complicated layout manager which provides flexible way of adding components in the container.

+ Applets:

+ Applet steps (on command line):

1. Write a java class inherited from java.awt.Applet class and override init() method. Write appropriate code in it.
2. To test applet with appletviewer tool, write comment before class (after import statements).

```
/*
<applet class="pkg.ClassName" width="200" height="200">
</applet>
*/3. Compile applet class.
javac -d . ClassName.java
```

4. Run applet into applet viewer:

```
appletviewer ClassName.java
```

5. To test applet in html file, write HTML file as follows:

```
<HTML>
    <HEAD>
        <TITLE>APPLET DEMO</TITLE>
    </HEAD>
    <BODY>
        <h2> Applet Sample </h2>
        <hr/>
        <applet code="pkg.ClassName.class" width="200" height="200">
        </applet>
    </BODY>
</HTML>
```

6. Open HTML file in the browser.

+ Applet:

- Applet is a java class which is loaded into the client browser and executed within client side JRE plugin of the browser.
- Applets are embedded into html web pages using <applet> tag having following attributes:
  - code="pkg.ClassName.class" i.e. name of the applet class with package.
  - codebase="dir path" i.e. path of directory in which applet package is kept. Required if applet package is in diffrent directory than the html page.
  - width="300" i.e. width of applet
  - height="300" i.e. height of applet

+ Applet Life Cycle:

- Applet class has five life cycle methods, which are overridden by the programmer as per requirement and called by the JRE/JVM within the client side browser.

1. public void init();

- When client requests a web page having embedded applet in it, the .class file is also loaded at client side. Client broser JVM get the info about applet class from <applet> tag (code attribute). It will load applet class into JVM, create its object, default constructor is executed and then init() method is invoked at runtime.
- This method is called only once in the life cycle of applet class object.
- Programmer should override this method to perform one time initialization like creating components on applet, etc.

2. public void destroy();

- When applet object is no longer required, browser JVM will garbage collect it.
- However before doing this, it will call destroy() method of the browser.
- Programmer should override this method, if applet is holding some resource like file or network connection and need to be released before object is destroyed.
- This method is called only once in the life cycle of applet object.

3. public void start();

- Whenever applet is made visible to end user, this method is invoked.
- Programmer should override this method to perform any operation that to be done when applet is shown to user e.g. initiating animation, if any.
- This method can be get called multiple times in the life-cycle of applet object.

4. public void stop();

- When user navigates from the applet web page, (the applet is no more shown to user) stop() method is called.
- Programmer should override this method to stop operations initiated into start() method, if any. i.e. stop animations.

5. public void paint();

- When whole applet or part of applet window need to be repainted, JVM invokes paint() method.
- Programmer should override this method to draw customized painting logic, if required.

+ Java IO:

+ File System Operations:

- File or directory path is represented by java.io.File object.
- This "File" class has several methods to deal with file system.
- boolean canRead();
- boolean canWrite();

- boolean canExecute();
- boolean setReadable(boolean perm);
- boolean setWritable(boolean perm);
- boolean setExecutable(boolean perm);
- boolean isHidden();

- boolean exists();
- boolean isFile();
- boolean isDirectory();

- boolean createNewFile();
- boolean mkdir();

- boolean delete();

- String[] list();
- long length();

+ Input and Output:

- Stream is java object in which data can be written or read from.
- There are two main types of streams:
  - Byte streams : Data is read/written byte by byte.
  - Character streams : Data is read/written char by char (unicode).

+ Byte Streams

- There are two types of Byte Streams:

1. OutputStream:
   - The data can be written into the output stream.
   - It is represented by abstract class: "java.io.OutputStream".
   - This class has following important methods:
     - abstract void write(int b);
       - Write single byte of data to the destination
     - int write(byte[] arr, int offset, int length);
       - Write part of byte array to the dest.
     - int write(byte[] arr);
       - Write whole array data to the dest.
     - void flush();
       - Write all data to the dest (if kept in mem).

- void close();
    - Close the stream and release resources.

2. **InputStream:**
    - The data can be read from input stream.
    - It is represented by abstract class: "java.io.InputStream".
    - This class has following important methods:
        - abstract int read();
            - Read single byte of data from destination
        - int read(byte[] arr, int offset, int length);
            - Read part of byte array from the dest.
        - int read(byte[] arr);
            - Read whole array data from the dest.
        - int available();
            - Returns num of bytes to be read.
        - void close();
            - Close the stream and release resources.

+ OutputStream heirarchy:

1. **FileOutputStream:**
    - Write the data to the file on the storage device (with help of OS).
2. **ObjectOutputStream:**
    - Converts java object and its info into the sequence of bytes (serialization).
3. **FilterOutputStream:**
    - Process data before writing it on underlying stream.
4. **DataOutputStream:**
    - Converts java primitive types into the object.
5. **BufferedOutputStream:**
    - Stores data in buffer (temp memory) before writing to underlying stream. Reduces number of write operations and thus improves performance.
6. **PrintStream:**
    - Convert java object data into proper (aligned) format. Neatly readable character data is produced
7. **ByteArrayOutputStream:**
    - Store data in form java byte array.

+ Demo of DataOutputStream / DataInputStream

Day 11:

+ Serialization:

- Converting java object into sequence of bytes is called as "serialization". These sequence of bytes contains data (contents) as well as metadata (info) of the object.

- Converting seqence of bytes back to the java object is called as "deserialization". It use metadata to recreate the object and restore its contents (using reflection).

- For serialization, the class must be inherited from marker interface "Serializable"; otherwise "NotSerializableException" will be thrown.

- Also class must have a public parameterless constructor.

- If you do not wish to serialize certain data member of the java object, it must be marked as "transient" (java keyword).

```
class Employee implements Serializable {
        private int empid;
        private String name;
        private double basicSalary;
        private transient double totalSalary;
        public Employee() {
                // ...
        }
        // ...
}
```

- Serialization & Deserilization example:

```
public static void writeStudent(Student s) throws Exception {
        FileOutputStream fout = new FileOutputStream("filepath");
        ObjectOutputStream out = new ObjectOutputStream(fout);
        out.writeObject(s);
        out.flush();
        out.close();
        fout.close();
}
public static void readStudent() throws Exception {
        FileInputStream fin = new FileInputStream("filepath");
        ObjectInputStream in = new ObjectInputStream(fin);
        Student s = (Student)in.readObject();
        System.out.println(s.toString());
```

```
            in.close();
            fin.close();
        }
```

+ Character Streams:

- Java stores characters as unicode (two bytes each).
- However char data can saved on disk using diffent encoding schemes e.g. ANSI, UTF-8, Unicode, Big Unicode, etc.
- The character streams are used for interconversion between java characters and encoding schemes.
- There are two main abstract classes:
  - java.io.Reader class
  - java.io.Writer class
- example:

```
        FileReader reader = new FileReader(path);
        System.out.println("ENCODING : " + reader.getEncoding());
        BufferedReader bufReader = new BufferedReader(reader);
        while (true) {
            String line = bufReader.readLine();
            if (line == null)
                break;
            System.out.print(line);
        }
        bufReader.close();
        reader.close();
```

+ Threading:

- Thread Creation in Java:
  1. extends Thread:
     - Programmer should write a class inherited from Thread class and override its "run()" method.
     - Object of that class should be created and its start() method is invoked.
     - example:

```
        class MyThread extends Thread {
            // ...
            public void run() {
                // ...
            }
        }
```

```
class Main {
        public static void main(String[] args) {
                MyThread t1 = new MyThread();
                t1.start();
        }
}
```

2. implements Runnable:

- Programmer should write a class inherited from "Runnable" interface and implement its "run()" method.
- Object of "Thread" class should be created and object of above created class should be passed to its constructor. Then call "start()" method of thread object.
- example:

```
class MyRunnable implements Runnable {
        // ...
        public void run() {
                // ...
        }
}
class Main {
        public static void main(String[] args) {
                MyRunnable r1 = new MyRunnable();
                Thread t1 = new Thread(r1);
                t1.start();
        }
}
```

- **Thread Life Cycle:**

1. New:
   - A new thread object is created.
   - Thread t1 = new Thread(...);
2. Ready to Run:
   - When start() method is invoked on thread object, the thread is associated with the scheduler.
   - Scheduler can select thread for the execution at any time based on scheduling algorithm.
3. Running:
   - When thread is selected for the execution by the scheduler, JVM invokes appropriate run() method and its execution begins.
4. Non-Runnable:

- Running thread may do IO request, sleep() or wait(). Because of that thread will go to non-runnable state. It is also called as "waiting" or "blocked" state.
- After IO completion, sleep() time completion or notify() done, thread may return back to "ready state" so that scheduler can schedule it again for further execution.
- However thread doing sleep() or wait() can be forcefully unblocked by calling interrupt() on the thread object (from another thread). This will raise InterruptedException into the sleeping/waiting thread.

5. Finally if run() method of a thread is completed, thread will be "dead" or "terminated".

+ Thread Join:

- When a thread calls join() method on some other thread object, it (calling thread) is paused until the given thread is terminated.
- If main thread calls t1.join(), then main thread will be paused until t1 is terminated.

+ Some important methods of thread class:

- void start();
- void setDaemon(boolean daemon);
- void join();
- void interrupt();
- static void sleep(long milli);
- void setName(String name);
- String getName();
- void setPriority(int pri);
- int getPriority();
- State getState();
- static Thread currentThread();
    - Get the "Thread" object in its run() method.
- static void yield();
    - Thread calling "Thread.yield()" method will goto ready state (from running) and thus JVM/scheduler gives CPU time for another thread.

+ Daemon threads:

- JAVA has mainly two types of threads:
    - non-daemon thread
    - daemon thread
- By default all threads are non-daemon threads.
- To make a thread as daemon, call setDaemon(true) on the thread object before starting that thread.

- When all non-daemon threads are terminated, process (in which JVM is running) is terminated. This will cause all daemon threads to terminate automatically.
- Thus daemon threads keep executing if at least one non-daemon thread is running.
- Daemon threads are also known as service threads, which provides service to non-daemon threads.
- The main thread is a non-daemon thread.

+ **Thread synchronization:**
- When multiple threads try to access same object at the same time, it is called as "race condition".
- This may result in corrupting object state and hence will get unexpected results.
- To avoid this Java provides sync primitive known as "monitor".
- Each java object is associated with a monitor object. The monitor can be locked or unlocked by the thread.
- If monitor is already locked, another thread trying lock it will be blocked until monitor is unlocked.
- The thread who had locked monitor, is said to be owner of monitor and only that thread can unlock the monitor.

- Java provides "synchronized" keyword to deal with monitors / synchronization.
- synchronized block:
  - example:

```
void fun() {
    // ...
    synchronized(obj) {
        // ...
    }
    // ...
}
```

- When certain part of the method should be executed by single thread at a time, we can use "synchronized" block.
- When a thread begin execution of the sync block, it will lock the monitor associated with given object (obj) and continue execute within that block.
- Meanwhile if another thread try to execute same sync block, it will try to lock the object monitor again.
- Since object is already locked, the second thread will be blocked, until first thread release the lock (at the end of sync block).
- When first thread release the lock, the waiting second thread will acquire it and continue to execute the block.
- **synchronized method:**
- When non-static sychronized method is invoked by the thread, it will lock the monitor associate with current object ("this"). When method completes, thread will release the lock.
- When static synchronized method is invoked by the thread, it will lock the monitor object associated with "ClassName.class" object.

- This ensure that two threads cannot execute same synchronized method on the same object.
- example:

```
class Account {
    private double balance;
    // ...
    public synchronized void withdraw(double amt) {
        // get cur balance from db
        // balance = balance - amt;
        // update new balance into db
    }
    public synchronized void deposit(double amt) {
        // get cur balance from db
        // balance = balance + amt;
        // update new balance into db
    }
    public double getBalance() {
        // get cur balance from db
        // return balance;
    }
}
```

- a1.withdraw(...) and a1.withdraw(...) executed by two different threads at the same time will not execute parallely. (As "a1" object is locked by thread1 and thread2 must wait for it).
- a1.withdraw(...) and a1.deposit(...) executed by two different threads at the same time will not execute parallely. (As "a1" object is locked by thread1 and thread2 must wait for it).
- a1.withdraw(...) and a2.withdraw(...) executed by two different threads at the same time can execute parallely. As lock is acquired by two threads on two different objects.
- a1.withdraw(...) and a2.getBalance(...) executed by two different threads at the same time can execute parallely. Because getBalance() method is not synchronized and hence the thread is neither trying lock the object nor will get blocked.
- If all methods in a class are synchronized, only one thread can perform any operation on object of that class. Such class is called as "synchronized" / "thread-safe" class. E.g. StringBuffer, Vector, Hashtable, etc.

Day 11:

==========

+ Sockets:

- Socket is an IPC mechanism used to communicate between two processes on the same machine or different machines (connected over network).
- Socket is defined as communication endpoint.
- Socket = IP address + port number
  - IP address : unique identifier for machine over network
  - port number: 16-bit logical identifier for the socket on a machine.
    - Range for port number : 0 to 65535
    - Ports 0 to 1023 are reserved for system use. Programmer should choose any other unused port.
- There are two main protocols:
  1. TCP:
     - Tranmission Control protocol
     - Connection Oriented
     - Acknowledgement is sent to transmitter
     - Reliable
     - Stream based
  2. UDP:
     - User datagram protocol
     - Connection less
     - No acknowledgement
     - Less Reliable
     - Packet based
- In java, UDP protocol communication is done using classes : DatagramSocket and DatagramPacket.
- In java, TCP protocol communication is done using classes : ServerSocket and Socket.

+ Remote Method Invocation:

- "Remote Procedure Call - RPC" is IPC mechanism.

- RMI demo : four coding steps:
  - **step1: MyIntf.java**

```
public interface MyIntf extends Remote {
    int sum(int a, int b) throws RemoteException;
    Student search(int roll) throws RemoteException;
}
```

  - Write a java interface containing methods to be called from client side.

- It must be inherited from marker interface "Remote".
- Each method in interface must throw "RemoteException".

- **step2: MyImpl.java**

```
public class MyImpl extends UnicastRemoteObject implements MyIntf {
    public MyImpl() throws RemoteException {
    }
    public int sum(int a, int b) throws RemoteException {
        return a + b;
    }
    public Student search(int roll) throws RemoteException {
        // ...
        return null;
    }
}
```

- Write java class to implement above remote interface and provide proper logic.
- This class must be inherited from UnicastRemoteObject.
- Class must have a paramless constructor throwing RemoteException.

- **step3: ServerMain.java**

```
public class ServerMain {
    public static void main(String[] args) throws Exception {
        MyImpl obj = new MyImpl();
        Naming.bind("abc", obj);
    }
}
```

- Create object of impl class and associate it with some name (friendly name).

- **step4: ClientMain.java**

```
public class ClientMain {
    public static void main(String[] args) throws Exception {
        MyIntf obj = (MyIntf)Naming.lookup("abc");
        int res = obj.sum(12, 5);
        System.out.println("Sum : " + res):
    }
}
```

- Find the object with name given at server side, type cast to interface ref and then call methods on it.

* Note:

Usually remote interface and other classes which are required for both client and server, are kept in the seperate package. The implementation class and server side main is kept in a package and finally client main class is kept into one more package.

* Note:

In order to execute server and client code from two different machines on the network, Client side Naming.lookup() method should contain complete URL of the server e.g. rmi://server-ip:1099/objname

Also in this case, server and common package should be deployed on the server machine; while client and common package should be deployed on client machine.

- Compilation and Execution of RMI code:
  - step5: Compile all above files:
    command: javac -d . *.java
  - step6: Run "rmiregistry" from the dir in which server package is kept (in command prompt 1).
    command: rmiregistry
  - step7: Run ServerMain from the dir in which server package is kept (in command prompt 2).
    command: java pkg.ServerMain
  - step8: Run ClientMain from the dir in which client package is kept (in command prompt 3).
    command: java pkg.ClientMain

- RMI:

- Calling java method from another JVM (process) on same or different machine (over network) is called as "RMI".
- Internally Stub and Skeleton objects are used to execute the method and receive result at the client side as shown in diag.

- Remote interface must be inherited from marker interface "java.rmi.Remote". It allows JVM to create stub for the given remote interface.
- Each method in remote interface and consructor in the implementation class must throw RemoteException. Since RMI internally performs networking, network connection may fail at runtime. So java forces each method to throw RemoteException, so that programmer can implement some logic to handle such exception.
- The implementation class must be inherited from "java.rmi.server.UnicastRemoteObject". The functionality of this class is internally used by the skeleton at the server-side.

- RMIREGISTRY:

- The info about impl class object (remote interface methods) is kept into rmiregistry associated with some "name". This is done by the Naming.bind() method at server side.
- When client program calls Naming.lookup(), it internally communicates with rmiregistry running on server and receive info about remote object. This info is used to create stub at runtime.

- Naming.lookup() returns pointer to stub object, on which client program can execute methods. These stub methods internally communicate with server skeleton to receive result.

- In the older java versions, stubs were not created at runtime. They should be created manually using "rmic" (RMI compiler) command.

- **RMI architecture:**

- Marshalling & Unmarshalling:

- Sending object contents over the network after serialization, is called as "marshalling".
- Receiving object contents from network (in form of byte sequence) and deserializing it, is called as "unmarshalling".

+ Java DataBase Connectivity - JDBC:

- JDBC Driver is a special program used to convert java requests to database understandable form and database response to java understandable form.
- There are four types of JDBC driver:

1. Type I driver:

- Also called as "JDBC ODBC Bridge Driver".
- ODBC drivers are implemented in C and can connect to older databases like Foxpro, Access, etc.
- Java has builtin JDBC driver to communicate with ODBC driver which in turn work with database.
- Name of driver class is sun.jdbc.odbc.JdbcOdbcDriver
- Due to multiple layers, it is slower.
- Also ODBC support is not fully present on all platforms.

2. Type II driver:

- Partially implemented in C and partially in Java.
- Use JNI to communicate betn Java and C layer.
- Better performance than Type I.
- Different driver for each database.
- Not truely portable (as partially in C).
- Outdated.
- e.g. Oracle OCI driver

3. Type III driver:

- JDBC driver communicate with a middleware, which in turn communicate with actual database.
- e.g. Weblogic RMI driver

4. Type IV driver:

- Completely implemented in Java and hence truely portable.
- Different for each database.

- Internally use sockets to communiate with database.
- e.g. Oracle Thin Driver.

+ JDBC is a specification:

- Specification is given in form of interfaces by "Sun Microsystem". Important Interfaces:
- Driver : Used to create and manage database connections.
- Connection : Encapsulate socket connection which communicate with database.
- Statement : Encapsulate SQL query to be executed on db.
- ResultSet : Represents database response to the query (rows and cols), used to fetch result row by row.
    - PreparedStatement : Used to execute parameterized query on the database.
    - CallableStatement : Used to execute stored procedure on the database.
- The specification given by Sun is implementeb by each JDBC driver. So internally each JDBC driver is set of classes inherited from above given interfaces. This ensures that same methods are present in all driver implementations.
- Thus JDBC program remains same irrespective of which JDBC driver or database you are using.

+ JDBC Example:

//0. adding JDBC driver into CLASSPATH

+ ON COMMAND LINE:
    - set CLASSPATH=/path/of/file/driver.jar;%CLASSPATH%
    - javac -d . ClassName.java
    - java pkg.ClassName
+ ON ECLIPSE:
    - Right click on project, Add New Folder. Give name "lib".
    - Copy driver jar file into "lib".
    - Right click on project, go to properties. Click "Java Build Path". Click "Libraries", then "Add Jars". Now select driver jar from "lib" directory within current project.

```
public class Demo46_Main1 {
	// oracle.jdbc.OracleDriver
	public static final String DB_DRIVER = "com.mysql.jdbc.Driver";
	// jdbc:oracle:thin:@adc:1521:oracle
	public static final String DB_URL = "jdbc:mysql://localhost:3306/test";
	public static final String DB_USER = "root";
	public static final String DB_PASS = "nilesh";
	public static void main(String[] args) throws Exception {
		//1. load and register jdbc driver
```

```
            Class.forName(DB_DRIVER);
            //2. create connection object
            Connection con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
            //3. create statement object
            Statement stmt = con.createStatement();
            //4. execute query and get result
            String sql = "SELECT DEPTNO, DNAME, LOC FROM DEPT";
            ResultSet rs = stmt.executeQuery(sql);
            //5. process the result
            while(rs.next()) {
                int deptno = rs.getInt("DEPTNO");
                String name = rs.getString("DNAME");
                String loc = rs.getString("LOC");
                System.out.printf("%d, %s, %s\n", deptno, name, loc);
            }
            //6. close connection, statement, ...
            rs.close();
            stmt.close();
            con.close();
        }
}
```

+ JDBC Insert Example:

```
public class Demo46_Main3 {
        public static final String DB_DRIVER = "com.mysql.jdbc.Driver";
        public static final String DB_URL = "jdbc:mysql://localhost:3306/test";
        public static final String DB_USER = "root";
        public static final String DB_PASS = "nilesh";
        public static void main(String[] args) throws Exception {
            //1. load and register jdbc driver
            Class.forName(DB_DRIVER);
            //2. create connection object
            Connection con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
            //3. create statement object
            Statement stmt = con.createStatement();
            //4. execute query and get result
            Scanner sc = new Scanner(System.in);
```

```
        System.out.println("enter deptno, name, loc : ");
        int deptno = sc.nextInt();
        String name = sc.next();
        String loc = sc.next();
        String sql = String.format("INSERT INTO DEPT VALUES(%d,'%s','%s')", deptno, name, loc);
        int count = stmt.executeUpdate(sql);
        System.out.printf("Records Inserted : %d\n", count);
        //5. process the result
        //6. close connection, statement, ...
        stmt.close();
        con.close();
    }
}
```

+ Using Statement:

```
        Class.forName("....");
        con = DriverManager.getConnection("...", "..", "..");
        Statement stmt = con.createStatement();
        int dno = sc.nextInt();
        rs = stmt.executeQuery("SELECT * FROM DEPT WHERE DEPTNO="+dno);
        while(rs.next()) {
            // ...
        }
        con.close();
```

+ Using PreparedStatement:

```
        Class.forName("....");
        con = DriverManager.getConnection("...", "..", "..");
        PreparedStatement stmt = con.prepareStatement("SELECT * FROM DEPT WHERE DEPTNO=?");
        int dno = sc.nextInt();
        stmt.setInt(1, dno);
        rs = stmt.executeQuery();
        while(rs.next()) {
            // ...
        }
        con.close();
```

Prasanna Wadekar 9970248407

+ Using CallableStatement : executing stored procedure

```
Class.forName(".....");
con = DriverManager.getConnection("", "", "");
CallableStatement stmt = con.prepareCall("{CALL sp_adddept(?,?,?)}");
dno = sc.nextInt();
stmt.setInt(1, dno);
name = sc.next();
stmt.setString(2, name);
loc = sc.next();
stmt.setString(3, loc);
stmt.execute();
```

+ DriverManager:

- Manual driver registration:

```
Class c = Class.forName("pkg.DriverClassName");
Driver drv = (Driver)c.newInstance();
DriverManager.registerDriver(drv);
```

- Connection Creation:

```
con = DriverManager.getConnection(DB_URL, ...);
```

DriverManager internally search for appropriate driver object (already registered) and then call its connect() method to create actual jdbc Connection and return it.

+ Scrollable ResultSet:

- While creating the statement, type and concurrency of result set can be specified.

- ResultSet types:

  - **ResultSet.TYPE_FORWARD_ONLY**
    - Only forward access, no reverse and random access.
    - This is default type.
  - **ResultSet.TYPE_SCROLL_INSENSITIVE**
    - Can perform forward, reverse and random access.
    - Any changes done in the database while resultset is in use, will not reflect into resultset.
  - **ResultSet.TYPE_SCROLL_SENSITIVE**
    - Can perform forward, reverse and random access.

- Any changes done in the database while resultset is in use, will reflect into resultset.

- **ResultSet functions:**
  - **To fetch records:**
    - boolean beforeFirst();
    - boolean afterLast();
    - boolean first();
    - boolean last();
    - boolean next();
    - boolean previous();
    - boolean absolute(int row);
    - boolean relative(int relRow);
  - **To check result set position:**
    - boolean isBeforeFirst();
    - boolean isAfterFirst();
    - boolean isFirst();
    - boolean isLast();

+ Updateable ResultSet:

- While creating the statement, type and concurrency of result set can be specified.
- concurrency types:

- ResultSet.CONCUR_READ_ONLY:

- default type
- using result set records can be only fetched from database.

- ResultSet.CONCUR_UPDATEABLE:

- using resultset records can fetched and also manipulated (insert, update, delete operations can be performed on the current record).
  - update related functions:
    - deleteRow();
      - delete current row from resultset and database
    - updateRow();
      - update changes made in current row into the database.
    - insertRow();
      - add newly created row into the database.

+ CachedRowSet:

- Inherited from ResultSet interface.
- Fetch all rows into the client memory, so that for accessing each row no need to connect to the database.
  - In this case connection can be closed while data is accessed at the client side (after receiving CachedRowSet). Due to this more number of clients can connect to database.

- However if large of number of records are fetched into client memory, performance will be reduced (due to shortage of memory).

+ Images in database:

// CREATE TABLE STUDENTS (ROLL INT PRIMARY KEY, NAME VARCHAR(20), MARKS DOUBLE, PHOTO BLOB);

- load and register driver
- get connection
- PreparedStatement stmt = con.prepareStatement("INSERT INTO STUDENTS(ROLL,PHOTO) VALUES (?,?)");

```
stmt.setInt(1, roll);
Blob bl = new BlobClassName();
OutputStream blobOut = bl.setBinaryStream(0);
FileInputStream fin = new FileInputStream("D:\\pic.jpg");
int b;
while((b=fin.read())!=-1)
        blobOut.write(b);
blobOut.flush();
fin.close();
blobOut.close();
stmt.setBlob(2, bl);
int cnt = stmt.executeUpdate();
```

+ Transactions:

- Transaction is set of SQL statements to be executed as a single unit. If all statements are executed successfully, then all of them should be commited to database. If any of the statement fails, then already executed statements should be rollbacked.

- example:

```
try {
        con.setAutoCommit(false);
        stmt1 = con.prepareStatement("UPDATE ...");
        stmt2 = con.prepareStatement("UPDATE ...")
        // ...
        stmt1.executeUpdate();
        stmt2.executeUpdate();
        con.commit();
} catch(Exception e) {
        con.rollback();
        // ...
} finally {
        // closing
}
```

Prasanna Wadekar
9970248407

37

+ **Java SE**
  - JVM (JRE) is on Desktop Machine
  - Console application, Windows application, Libraries (jars)

+ **Java Enterprise Edition**
  - Create Enterprise applications
  - Typically n-tier applications
  - Is a specification given by Sun Microsystems?
  - It is implemented by the web servers and application servers
  - Specification includes:
    - Servlet, JSP & Java Beans, JSF, EJB, JPA
  - Java EE also has many Third Party Frameworks:
    - Struts, Spring, Hibernate, etc.
  - **Web Server**
  - Application/sw that allow to execute one or more web applications in it, which can be accessed over the network (intranet or internet)
  - Java Web Server = Web Container + Extra Services
    - Web Container = Servlet Engine + JSP Engine + etc.
    - Extra Services = JNDI + Connection Pooling + etc.
  - Examples:
    - Apache Tomcat, Lotus Domino, etc.
  - **Application Server**
    - is more than a web server; serves business logic.
    - Java Application Server = Web Container + EJB Container + Extra Services
    - Examples:
    - Sun Glassfish, JBOSS, WebSphere, WebLogic, Pramati, Enhydra, Orion, etc.

PRASANNA

  - **Apache Tomcat - Web Server**
    |- bin - scripts to start or stop web server & tomcat executables
    |- conf - contains xml files for conf web server e.g. port configuration
    |- lib - contains lot of jar files implementing java ee specs
    |- logs - for maintaining logs
    |- temp - for temp files
    |- webapps - hot deployment directory - where the web applications are deployed
    |- work - contains intermediate files created during execution of the application (useful for JSP Execution)

+ **Web Application**
  - Collection of web pages and resources.
  - Web pages can be static (HTML) or dynamic (processed at server)

+ **HTTP protocol**
  - connection-less, state-less
  - request response model

- **HTTP Request**
  - From Browser to Web Server
  - http://servername:port/application/webpage
  - Request Contents:
    - Server Name (IP) & Port Num
    - Resource Identifier (/application/webpage)
    - Request Method - GET / POST
    - Request length
    - Request Headers (info about the client)
    - Cookies
    - Request Body

- **HTTP Response**
  - From Web Server to Browser
  - Response Contents:
    - Resp Status code e.g. 403, 404, 500, 200, etc.
    - Content/MIME Type e.g. text/html, text/xml, text/plain, image/png, image/jpg, image/gif, audio/mp3, video/mpeg, etc.
    - Response length
    - Response headers (info about the server)
    - Cookies
    - Response Body

- **HTTP Request Methods:**
  - **GET:**
    - data is sent via url (querystring)
    - not secure
    - send limited data
    - faster
  - **POST:**
    - data is sent via req body
    - secure
    - no data limit
    - slower
  - **HEAD:**
    - resp only contain headers not body
  - **PUT:**
    - for upload/put a file on web server
  - **DELETE:**
    - for delete a file from web server
  - **TRACE:**
    - for dubugging/tracing http req/resp data
  - **OPTIONS:**

- to know which methods are supported on server side

+ JavaEE WebApplication Directory Structure e.g. myweb:
- [myweb]
  - |- *.html, *.htm, *.jsp, ...
  - |- *.jpg, *.png, ...
  - |- [WEB-INF]
    - |- web.xml
    - |- [classes]
      - |- *.class in pkg
    - |- [lib]
      - |- *.jar third party jars
    - |- [src]
      - |- *.java files

+ Servlet:

- Servlet is a java class, which is executed within web container when client makes request and generate response which is sent to the client.

The servlet class must be directly or indirectly inherited from Servlet interface.

- javax.servlet.Servlet interface
  + void init(ServletConfig config) throws ServletException;
    - For the first request from the client, web container load the servlet class in memory and create its object
    - Immdiately after creating the object, it call its init() method (only once in life cycle of servlet object)
    - Programmer should override this method for performing one time initialization.
    - ServletConfig object (param) is used to access servlet config info from web.xml file e.g. init params
    - If there is any exception/error during initialization, you must throw Servlet Exception.

  + void destroy();
    - When servlet is object is no longer used or web container is going down, then it calls destroy() method before destroying servlet object. (only once in life cycle of servlet object)
    - After call to this method, object is ready for garbage collection.
    - Programmer should override this method for performing deinitialization task.

  + void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException;
    - For each request web container call the service() method.
    - Programmer should override this method to handle the request, process it and generate response.

- after method is executed, web server will send the generated resp to the client.
- the ServletRequest and ServletResponse objects are created by the web container before calling this method.

- javax.servlet.GenericServlet class
  - this class implements basic functionality of protocol-independent servlet
  - implements all methods from "Servlet" interface except service() method, hence this class is abstract class.

- javax.servlet.http.HttpServlet class
- this class implement basic functionality of the servlet which will be accessed using http protocol.
- This class has methods corresponding to each req method.
i.e. doGet(), doPost(), doHead(), doPut(), ...
- void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException;
- void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, **IOException;**
  - HttpServlet implement service() method and within that method, it checks the req method type and call corresponding doXXXX() method.

+ Servlet Example:

```
package pkg;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
	public void init(ServletConfig config) throws ServletException {
		super.init(config);
		// ...
	}
	public void destroy() {
		// ...
	}
	public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
	ServletException, IOException {
		processRequest(req,resp);
	}
	public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
	 ServletException, IOException {
		processRequest(req,resp);
	}
	public void processRequest(HttpServletRequest req, HttpServletResponse resp) throws
	ServletException, IOException {
```

```
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println("<html>\n<body>");
                out.println("<h2>Hello Servlet</h2>");
                out.println("</body>\n</html>");
        }
}
```

+ Servlet declaration in web.xml:

+ simple declaration:

```
<servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>pkg.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

+ Alternatively, servlet can be declared using annotation @WebServlet.

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
        // ..
}
```

+ <load-on-startup>

- By default servlet class is loaded and instantiated, when first request is arrived for the servlet.
- Using <load-on-startup> under <servlet> tag, servlet can be loaded and instantiated as soon as application is deployed.
- e.g. <load-on-startup>1</load-on-startup>
- The number given in body of tag, indicate sequence of servlet loading, if there are multiple servlets with load-on-startup tag.

+ <init-param>

- Some configurable information can be associated with servlet using init-param under <servlet> tag in web.xml.
- Example:

```
<init-param>
        <param-name>DB_URL</param-name>
        <param-value>jdbc:mysql://localhost:3306/test</param-value>
</init-param>
```

- This value can be accessed in servlet class using ServletConfig object as:

```
ServletConfig config = this.getServletConfig();
```

```
String db_url = config.getInitParameter("DB_URL");
// ...
```

+ State Management:

+ Client Side State Management:
  - the state/info of client is stored on client machine
  - less memory is needed at the server side
  - less secure, client can access and/or modify
  - Ways: Cookie, QueryString, Hidden Form Fields

+ Server Side State Management:
  - the state/info of client is stored on server machine
  - large memory is needed at the server side
  - more secure, client cannot directly access
  - Ways: Session, Application

+ Cookie:

- To store info about the client, server send that info in textual key-value form to the client known as "cookie".
- Once cookie is received by the client, thereafter with each request cookie is sent back to the server.
- Cookie can store only text data upto 4KB.
- **Cookies Types:**
  - **Temporary Cookie:**
  - cookie is stored in browser memory and will be destroyed when browser is closed.
  - **Persistent Cookie:**
  - cookie is stored in client machine (disk) as text file and will be persisted even if browser is closed.

- **To create and send cookie:**

```
Cookie c = new Cookie("key", "value");
// c.setMaxAge(secs); // persistent
resp.addCookie(c);
```

- **To receive cookie and get data:**

```
Cookie[] arr = req.getCookies();
for(Cookie c : arr) {
        if(c.getName().equals("key")) {
                String value = c.getValue();
                // ...
        }
}
```

- **Cookie methods:**

```
Cookie(String name, String value);
String getName();
String getValue();
```

- **Session:**
  - For each client one session object is created on the server side, in which client specific state/info can be saved.
  - Session object is like a map, where data is stored in key/value pairs.
  - The key must be a string, while value can be any object.

  - **HttpSession methods:**

```
void setAttribute(String key, Object value);
Object getAttribute(String key);
void invalidate();
boolean isNew();
String getId();
```

  - **To get session object:**

```
HttpSession session = req.getSession();
```

  - to change session timeout:- web.xml in <web-app>

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

  - **Session Tracking:**
    - Each session is identified using a unique session id, which is associated with the client.
    - There are two ways of this association (tracking):
      - **using sessionid cookie:**
        - by default, when new session is created (req.getSession() is called first time) a cookie is created and sessionid is sent to client via that cookie.
        - for suqsequent calls to req.getSession() access the appropriate session object by getting sessionid from that cookie.
      - **using url rewriting:**
        - In case cookies disabled, sessionid can be maintained using url.
        - resp.encodeURL() and resp.encodeRedirectURL() methods are used to embed sessionid into the url.
          e.g. http://server:port/app/page;jsessionid=374334

+ **ServletContext : (application)**
  - For each web application, web container creates a singleton object called as "ServletContext" during application deployment.
  - This object is used

- To store the data globally so that it can be accessed for all request to all pages (servlets) by all users.
- To navigate from one page (servlet) to another using RequestDispatcher.
- in servlet / config class:

```
ServletContext getServletContext();
```

- To save and retrieve data from the ServletContext:

```
void setAttribute(String key, Object value);
Object getAttribute(String key);
```

+ Context Parameter:

+ ServletConfig can be used to get init param for specific servlet (from web.xml).
+ In web.xml we can declare the values which can be accessible in entire application in form of context params:

```
<web-app>
    // ...
    <context-param>
        <param-name>color</param-name>
        <param-value>yellow</param-value>
    </context-param>
    // ...
</web-app>
```

+ This param can be accessed into the web application via ServletContext object as:

```
String colorValue = context.getInitParameter("color");
```

+ Listeners:

- These interfaces are used to handle events in the web application e.g. application start (context initialization), application stop (context uninit), session start, session end, new value added into session, any value removed from session, etc.

- interface ServletContextListener:

```
void contextInitialized(ServletContextEvent e);
void contextDestroyed(ServletContextEvent e);
```

- interface HttpSessionListener:

```
void sessionCreated(HttpSessionEvent e);
void sessionDestroyed(HttpSessionEvent e);
```

- interface HttpSessionBindingListener:

- interface HttpSessionAttributeListener:
    - void attributeAdded(HttpSessionBindingEvent e);
    - void attributeRemoved(HttpSessionBindingEvent e);
    - void attributeReplaced(HttpSessionBindingEvent e);

- **How to use listeners?**
  - step1: write a class implementing required listener
  - step2: write the appropriate logic into appropriate method
  - step3: inform web container about this listener in web.xml

- example:

```
public class Global implements ServletContextListener, HttpSessionListener {
        ServletContext ctx;
        public void contextInitialized(ServletContextEvent e){
                System.out.println("contextInitialized");
                ctx = e.getServletContext();
                ctx.setAttribte("hits", new Integer(0));
        }
        public void contextDestroyed(ServletContextEvent e){
        }
        public void sessionCreated(HttpSessionEvent e){
                System.out.println("sessionCreated");
                Integer count = (Integer)ctx.getAttribute("hits");
                count = count + 1;
                ctx.setAttribute("hits", count);
        }
        public void sessionDestroyed(HttpSessionEvent e){
        }
}
```

- **Declare listener in web.xml**

```
In web.xml -> <web-app>
<listener>
        <listener-class>sun.krd.bkshop.Global</listener-class>
</listener>
```

- Alternatively listener can be declared using annotation @WebListener

```
@WebListener
public class Global implements ServletContextListener, HttpSessionListener {
        // ...
}
```

+ **Navigation:**
  + Redirection:
    - resp.sendRedirect(url);
    - When sendRedirect() is called, a temp response (status code 302 and destination url) is sent to the browser; due to which browser make a new request to the new link.

- In this case two requests are originated from the browser and hence browser is aware of the navigation.
- This is slower process.
- Can redirect from any page to any other page of the same application or different application.

+ RequestDispatcher:

- RequestDispatcher rd = req.getRequestDispatcher("url");

OR RequestDispatcher rd = ctx.getRequestDispatcher("/url");

- rd.forward(req, resp);

OR rd.include(req, resp);

- Only one request is originated from the client, and single response is given back.
- This is faster.
- Browser is not aware of the navigation.
- Navigation can be done only to the pages within the same application.
- The req object can be used to carry extra info using:

```
void setAttribute(String key, Object value);
Object getAttribute(String key);
```

+ Forwarding:

- Request is forwarded to next servlet from which response will be given to the client.

+ Including:

- Request is given to next servlet, which performs some processing and return back to the calling servlet. The response generated by the second servlet will be included into first servlet's response.

+ JSP:

- Servlet = Business Logic (Java) + Presentation Logic (HTML)
- JSP = Presentation Logic (HTML) + Business Logic (Java)
- JSP => Servlet

+ Stages for compilation and execution of JSP:

---------- JSP Engine ----------

1. Translation Stage:

- When first request is made for the JSP page, it will be loaded by the web container inside JSP engine.
- JSP engine translate the JSP page into a servlet's java code. This .java file can be found in tomcat's "work" folder.
- If there is any error in JSP syntax (e.g. scriptlets), then this stage fails.

2. Compilation Stage:

- The translated servlet's .java file will be compiled into a .class file at runtime.
- If there is any java syntax error, then this stage fails.

---------- Servlet Engine ----------

3. Instantiation (Loading) Stage:

- The .class file will be loaded and object of the translated servlet will be created.
- Immediately after this init method of the JSP i.e. jspInit() will be executed.
- If this method throws any exception, this stage fails.
- This stage is also called as Loading or Initialization stage.

4. Request Handling Stage:

- All above stages are done only for the first request of the JSP file; However this stage is executed for each request.
- For each request, _jspService() method is executed (which is made up of all the scriptlet and expressions in the JSP file).

5. Destruction Stage:

- When servlet object is no longer used or web container is going down, jspDestroy() method will be executed after which servlet object will be garbage collected.

+ JSP syntax:

1. directive <%@ .... %>

- <%@page language="java" import="java.util.*, java.io.*" ..%>
  - mainly controls servlet translation
- <%@include file="file.ext" %>
  - adding external html or jsp file statically.
- <%@taglib .... %>
  - used for custom and third-party tags

2. declaration <%! .... %>

- used to declare fields and methods which will not be executed per request. e.g. jspInit(), jspDestroy(), other methods, all fields, etc.
- can write one or more declaration blocks

3. scriptlet <% .... %>

- used to write java statements to be executed per request.
- all code written here will be part of _jspService() method during translation phase.

4. expression <%= .... %>

- the java expressions whose result (string) will be directly added into generated html.

+ Implicit JSP objects:

- Few objects are directly accessible in request handling stage (without need of their creation) - in scriptlets and expressions - called as "implicit objects".

1. request: HttpServletRequest
2. response: HttpServletResponse

3. config: ServletConfig
4. application: ServletContext
5. session: HttpSession
6. out: JspWriter
7. page: Object -> this pointer
8. pageContext: PageContext -> to save some data on page scope (accessible on that page for that request only)
9. exception: Throwable -> accessible only in error pages

+ First JSP Example:

```
import java.util.*;
class FirstServlet extends HttpServlet {
        List<String> list = new ArrayList<String>();
        void init(...) ... {
                list.add("nilesh");
                list.add("rahul");
                list.add("sandeep");
                System.out.println("init() called");
        }
        void destroy() {
                System.out.println("destroy() called");
        }
        void doGet(...) {       process(...);   }
        void doPost(...) {      process(...);   }
        void process(...) {
                System.out.println("process() called");
                resp.setContentType("text/html");
                PrintWriter out = resp.getWriter();
                out.println("<html>...<body>");
                out.println("<h2>JSP Demo</h2>");
                for(String str : list)
                        out.println(str + "<br/>");
                out.println("</body>...</html>");
        }
}
```

+ @page directive:

- language="java"
- contentType="text/html"
  - will be converted to resp.setContentType("text/html");
- import="packagename"

- session="true"

- true: internally req.getSession() will be executed.
- false: session will be null.

- buffer="8kb" autoFlush="true"
  - all resp generated will be stored in a temp buffer before resp is sent to the client.
  - autoFlush=true : when resp is generated or buffer is full it will be automatically flushed to the client.
    - autoFlush=false : the contents will be immediately flushed to the client.
  - **isThreadSafe="true"**
  - true: single thread will be used service requests. Due to which thread sync problems may not occur.
  - false: for concurrent requests multiple threads will be created and multiple servlet objects will be created. This reduces performance.
    - internally generated servlet implement "SingleThreadModel" marker interface, which is deprecated with newer versions.
  - **isErrorPage="false"**
    - false: simple jsp page
    - true: this page is used to display some error to the end user (custom error page). This page will have access to "exception" object.
  - **errorPage="myerr.jsp"**
    - this attr is used in normal jsp page (not in err page)
    - if current page get some error, web container will redirect to the given error page.
  - The error pages are used to display well-formatted error messages. The error pages can be associated with some error codes in web.xml file as follows:

```
<web-app>
    // ...
    <error-page>
        <error-code>404</error-code>
        <location>/not.jsp</location>
    </error-page>
</web-app>
```

+ The JSP having minimum scriptlets is considered as better JSP. So programmers should avoid writing scriptlets wherever possible.
This can be achieved with help of using standard tags (actions), third party tags or custom tags.

+ JSP standard actions:
- JSP built-in tags used to specific tasks.

1. jsp:plugin & jsp:fallback
   - these tags are used to embed applets into jsp pages.
   example:

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="/application" width="200" height="300">
    <jsp:fallback>Applet Not Loaded</jsp:fallback>
</jsp:plugin>
```

- This code will be internally converted into html's <object> or <embed> tag.

2. jsp:forward, jsp:include and jsp:param

```
<jsp:forward page="pageurl"/>
```

OR

```
<jsp:include page="pageurl"/>
```

- Will get converted into .....

```
RequestDispatched rd = req.getRequestDispatcher("pageurl");
rd.forward(req, resp);          OR          rd.include(req, resp);
```

* jsp:include is used to for dynamic inclusion of the file.

```
<jsp:forward page="page2.jsp">
    <jsp:param name="key1" value="value1"/>
</jsp:forward>
```

So that in next page page2.jsp,

```
String val = request.getParameter("key1");
```

3. jsp:useBean, jsp:setProperty, jsp:getProperty

+ JSP and Java Beans

+ Java Bean:

- Simple java class having fields, param less constructor, getters/setters, one or more business logic methods

```
// CapBean.java
// java bean class
public class CapBean {
    // fields
    private String word;
    // param less ctor
    public CapBean() {
        this.word = "";
    }
    // getters/setters
    public void setWord(String word) {
        this.word = word;
    }
    public String getWord() {
        return this.word;
    }
```

```
        // business logic method
        public String getCapital() {
                String res = word.toUpperCase();
                return res;
        }
}

// input.jsp
<form method="post" action="out.jsp">
        Word : <input type="text" name="in_word"/>
        <input type="submit" value="Submit"/>
</form>

// output.jsp
<!-- create obj of bean -->
<jsp:useBean id="cb" class="pkg.CapBean" scope="page"/>

<!-- call setters to set property -->
<jsp:setProperty name="cb" property="word" param="in_word"/>

<!-- call getters to get property -->
Word : <jsp:getProperty name="cb" property="word"/>

<!-- call business logic method -->
Result : <%= cb.getCapital() %>
Result : <jsp:getProperty name="cb" property="capital"/>
```

+ **Rules to write Java Beans:**
  - Write a simple java class having following members:
    1. one or more fields
    2. param less constructor
    3. getter/setter methods (proper camel case)
    4. one or more business logic method (any name)

+ Standard actions for java beans:
  1. <jsp:useBean id="obj" class="pkg.ClassName" scope="page"/>
  - This tag check the whether any object is present in the given scope with given name (id).
    - If object is present, it will be accessed by that name.
    - If object is not present, class will be loaded and object will be created at runtime.
    - Then object will added into given scope with given name.

- **Java Bean scopes:**
  i. application: bean is stored into ServletContext and hence accessible in all requests to all pages by all users

ii. session: bean is stored into HttpSession and hence accessible in all requests to all pages by current user.

iii. request: bean is stored into HttpServletRequest and hence accessible in all the pages on which request is forwarded or included.

iv. page: bean is stored into PageContext and hence accessible in current request to current page.

2. **<jsp:setProperty name="obj" property="propName" value="fix_value"/>**
<jsp:setProperty name="obj" property="propName" param="req_param"/>
- Internally find corresponding setter method (i.e. setPropName()) and execute it on given object.
- "value" is used to give fix value, while "param" is used to give value from request parameter.

3. **<jsp:getProperty name="obj" property="propName"/>**
- Internally find corresponding getter method (i.e. getPropName()) and execute it. The return value will be added into html response.

* Before calling <jsp:setProperty> or <jsp:getProperty> the jsp page must have <jsp:useBean> tag.

+ Model I architecture:
- Also called as "Model-View" Architecture.
- View -> Presentation Logic -> JSP pages
- Model -> Business Logic -> Java Beans
- used for very small web applications

+ Model II architecture:
- Also called as "Model-View-Controller" Architecture.
- e.g. Third party frameworks like struts, spring, etc.

+ JSP Expression Language:
- Used to access (read) values from different scopes without using scriptlet or expression syntax.
- EL syntax:
  ${scopeName.variableName}
- Four scopes:
  - pageScope, requestScope, sessionScope, applicationScope
  e.g. ${sessionScope.myname}
- This syntax can be further simplified by omitting scopeName.
  i.e. ${varName}

In this case, the var will be searched from the lowest scope to highest scope.
- Using this you can call getter methods of the object as follows:
  ${scopeName.objName.propName}

e.g.${sessionScope.lb.username}
- Using EL syntax you can also call any method on the object
  ${scopeName.objName.method()}
- If any EL expression is not found, it will be ignored (no error will be raised).
- You can write math expressions (of consts) into EL
  ${3 + 4 / 2 * 7}

- EL can be used to access values from the special implicit variables (objects).
  EL implicit objects:
  1. param
     - ${param.varName}
       -> req.getParamter("varName");
  2. paramValues
     - ${paramValues.varName}
       -> req.getParamterValues("varName");
  3. header
     - ${header.varName}
       -> req.getHeader("varName");
  4. headerValues
  5. initParam
     - ${initParam.varName}
       -> config.getInitParameter("varName");
  6. cookies
     - ${cookies.cookieName}
       -> will get the value of the cookie with cookieName
  7. pageContext

**** All standard actions and EL syntax must be used outside scriptlets <% ... %>**

*** All EL in the page can be ignored using
<%@ page isELIgnored="true" %>

+ JSP Tags - business logic embedded into presentation logic
+ Standard Actions:
  - Built-in tags with prefix "jsp:"
  - e.g. forward, include, plugin, fallback, beans related
+ **Third Party Tag Libraries:**
  - Struts Tag Library, Spring Tag Library, JSTL, etc.
  - JSTL - JSP Standard Tag Library - Sun Microsystems
    - Basic prog constructs: switch, if-else, loops, etc.
    - JDBC tags, etc.
+ **Custom Tags:**
  - Programmer can define his own tags for special needs

+ JSTL:
- **To use JSTL in the JSP web page:**
  - **step1: add appropriate jars into WEB-INF/lib directory**
    - standard.jar, jstl.jar
  - **step2: at the start of the JSP page use "taglib" directive**
    - <%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt" %>
    - * prefix = used to avoid tag name clashes

* uri = identifier for tags collection

- **step3: use the apprpriate tag with proper attributes into the JSP page. example:**

```
books.jsp
-----------
<c:forEach var="b" items="${bb.bookList}">
        <input type="checkbox" name="book" value="${b.bookid}"> ${b.name}
</c:forEach>

validate.jsp
-------------
<c:choose>
        <c:when test="${lb.status=='true'}">
                //...
        </c:when>
        <c:otherwise>
                //...
        </c:otherwise>
</c:choose>
```

+ Custom Tags:

+ JSP has mainly two types of tags:
  - classic tags and simple tags
  - both are inherited from marker interface "JspTag".
  - To implement these tags you can use interfaces "Tag" and "SimpleTag" or adapter classes "TagSupport" and "SimpleTagSupport" respectively.

+ Steps to implement custom (Simple) tag:

- step 0: decide the application, name, attributes and body of the tag.
  e.g. <my:wish uname="some_name"/>
  This tag should print greeting meesage for the given name.

- **step 1:** write the tag handler class inherited from "SimpleTagSupport" e.g. WishTag. Also add param less ctor.
  - in the class write number of fields equal to number of attributes, with names matching to attribute. Also add getter/setter for them.
  - override doTag() method of the "SimpleTagSupport" class and implement the business + presentation logic in it.

```
public class WishTag extends SimpleTagSupport {
        private String uname;
        public WishTag() {
                this.uname = "";
        }
```

```
        public void setUname(String uname) {
                this.uname = uname;
        }
        public String getUname() {
                return this.uname;
        }
        public void doTag() {
                JspContext ctx = this.getJspContext();
                JspWriter out = ctx.getOut();
                out.println("Hello, " + uname);
        }
}
```

- **step 2:**
  - write tag library descriptor (tld) xml file inside WEB-INF to give complete information about the tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
jsptaglibrary_2_1.xsd">
    <tlib-version>1.0</tlib-version>
    <short-name>custom</short-name>
    <uri>/WEB-INF/custtags.tld</uri>
    <tag>
        <name>wish</name>
        <tag-class>sub.krd.bkshop.WishTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>uname</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.String</type>
        </attribute>
    </tag>
</taglib>
```

- step3: use the tag into the JSP page
  - <%@taglib prefix="my" uri="/WEB-INF/custtags.tld" %>
  - use in page

```
<my:wish uname="${lb.username}"/>
```

+ SimpleTag life cycle:

1. When page containing custom tag is accessed first time, during translation stage:
   - the tld file is referred (via given prefix in @taglib)
   - from tld used tag is found and validated (syntax)
2. For each time tag is used, tag-class (given in .tld file) is loaded and object is created at runtime. Paramless constructor will be called here.
3. setJspContext() method will be called by the container and current JSP's PageContext object will be passed into it. This object contains all info needed to process JSP page.
4. If tag is child of any other tag, then setParent() method will be called.
5. Then container calls setter methods for all attributes used in the JSP file during tag invocation.
6. If tag has some body, then setJspBody() method will be called to provide tag body.
7. Finally container calls doTag() method of the tag class, which does server side processing and generate html output if any.
8. After doTag() is completed, the tag's generated html will be added into page response.

* If not overridden, setJspContext() method of the "SimpleTagSupport" will be called (step 3), which will save the current JspContext so that it can be accessed via call to getJspContext() [usually in doTag() method].

** Refer docs of "interface Tag" for classic tag lifecycle.

+ **Hibernate:**
  - Hibernate is an ORM tool.
  - ORM - Object Relational Mapping
  - Java Object <= mapping => Relational Database
    - Java Class (Entity) - Database Table
    - Class Fields - Table Columns
  - **SessionFactory - hibernate.cfg.xml (src directory)**
    - contains full info about the db to be connected.
    - also contains info about entity classes
    - In an application there must be single object of SessionFactory
  - **Session -**
    - Encapsulate JDBC connection
    - helps transaction management
    - track the changes into entity objects and update in db
    - created from SessionFactory

* SessionFactory can get database connection by using DriverManager (same as JDBC), DataSource or any other way.

+ **Steps to use hibernate into the project:**
  - step1: copy all hibernate jar files into WEB-INF/lib folder along with JDBC driver jar.
  - step2: create a new xml file into src directory i.e. hibernate.cfg.xml
    - This file contains info about database or data-source

- This file also contains info about all entity classes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
 <session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/dacdb</property>
  <property name="hibernate.connection.username">nilesh</property>
  <property name="hibernate.connection.password">nilesh</property>
  <property name="hibernate.connection.pool_size">2</property>
  <property name="hibernate.current_session_context_class">thread</property>
  <property name="hibernate.connection.autocommit">true</property>
  <property name="hibernate.show_sql">true</property>
  <mapping class="sun.krd.hbshop.Login"/>
 </session-factory>
</hibernate-configuration>
```

- **step3**: create an helper class, say HbUtil, to initialize hibernate SessionFactory object.

```
public class HbUtil {
	public static SessionFactory factory;
	static {
		Configuration configuration = new Configuration().configure();
		StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder(). ,
		applySettings(configuration.getProperties());
		factory = configuration.buildSessionFactory(builder.build());
		System.out.println("Hibernate Session Factory created ...");
	}
}
```

- step4: create one or more entity classes using annotations from javax.persistence package. Also make their entries into hibernate.cfg.xml.

```
		@Entity
		@Table(name="BOOKS")
		public class Book implements Serializable {
			@Id
			@Column(name="BOOKID")
			private int id;
			private String name;
			private String author;
			private String subject;
```

```
        private double price;
        // paramless constructor
        // getters/setters
    }
```

- step5: To perform db operation into servlet, JSP, DAO or any other class, create object of hibernate "Session" and call its methods. Finally close the session.

+ Hibernate Db Operations:

1. Insert Operation - Book b:

```
Session hbSession = HbUtil.factory.openSession();
hbSession.save(b);          // persist(b);
hbSession.flush();
hbSession.close();
```

2. Delete Operation - id=23:

```
Book b = new Book();
b.setBookid(id);
Session hbSession = HbUtil.factory.openSession();
hbSession.delete(b);
hbSession.flush();
hbSession.close();
```

3. Update Operation - [name,subject,price,author -> id]

```
Book b = new Book(id,name,author,subject,price);
Session hbSession = HbUtil.factory.openSession();
hbSession.update(b);
hbSession.flush();
hbSession.close();
```

4. Select by PK Operation - id

```
Session hbSession = HbUtil.factory.openSession();
Book b = hbSession.get(Book.class, id);
hbSession.close();
```

+ Hibernate Entity Life Cycle:

1. Transient:

- The entity objects created in java application and hibernate is not aware of them.

2. Persistent:

- The objects which are part of hibernate cache are in "persistent" state; where any changes in those objects will be automatically reflected into db.
- When any transient/detached objects are added into cache using save() or update() method, they become persistent.
- Also all objects created by session via get() or list() method are persistent.

3. Detached:

- Any object removed from the hibernate cache is in this state
- The evict() operation will detach a particular object; while clear() will detach all objects in the cache.

4. Removed:

- If db row corresponding to any object is deleted then object is in this state.
- Any persistent object doing delete() operation will come to this state.

+ Hibernate Relations:

- Hibernate allows association among hibernate entities as follows:
  - One to One
  - One to Many
  - Many to One
  - Many to Many
- It facilitates fetching related objects from the database easily.
- Example of One to Many relationship:

```
// Dept.java
@Entity
@Table(name="DEPT")
public class Dept {
	@Id
	private int deptno;
	private String dname;
	private String loc;

	@OneToMany(fetch=FetchType.LAZY, mappedBy="deptno")
	private List<Emp> empList;

	// constructor and getter/setters
}

// Emp.java
@Entity
@Table(name="EMP")
public class Emp {
	private int empno;
	private String name;
	private double salary;
	private int deptno;

	// constructor and getter/setters
}

// Main.java
```

```
public static void main(String[] args) {
        Session session = HbUtil.getFactory().openSession();
        Dept d =  (Dept) session.get(Dept.class, 1);
        System.out.println("DEPT : " + d);
        List<Emp> list = d.getEmpList();
        for (Emp e : list) {
                System.out.println(e);
        }
        session.close();
}
```

- Hibernate relations support two types of fetching:

1. Lazy Fetching:

- Related objects (child table data) are fetched only when they need access.
- Typically fires two different queries. First on master table while fetching its data and second on child table when element is accessed.
- In above example, session.get(Dept.class, 1) - fires:
  select DEPTNO, DNAME, LOC from DEPT where DEPTNO=?
- And, d.getEmpList() fires:
  select EMPNO, ENAME, SALARY, DEPTNO from EMP where DEPTNO=?

2. Eager Fetching:

- Related objects (child table data) are fetched along with the main object (master table data).

- Internally use single join query to fetch the results from both tables.
- In above example, if fetch type is changed to FetchType.EAGER then, session.get(Dept.class, 1) - fires:

select d.DEPTNO, d.LOC, d.DNAME, e.DEPTNO, e.EMPNO, e.ENAME, e.SALARY from DEPT d left outer join EMP e on d.DEPTNO=d.DEPTNO where d.DEPTNO=?

+ Struts2:

+ Steps for First Struts Application:

1. Create "Dynamic Web Application" - mystruts
2. Add struts 2 jar files into WEB-INF/lib folder
3. In "src", create "struts.xml" file (with proper grammer).
4. In web.xml, make entry of the struts2 controller filter using <filter> element.
5. create "in.jsp" -> form action="caps" - textfield(intext), submit - using struts (third party) tags
6. create "CapAction" (model) class extends ActionSupport
   fields: intext, outtext; constructor; getter/setter;
   execute() method - business logic (return a string "res")
7. create "out.jsp" -> display intext & outtext using struts tags
8. modify struts.xml
   <action name="caps" class="pkg.CapAction">

```
        <result name="res">/out.jsp</result>
    </action>
```

+ **Model II Architecture:**
  - Model View Controller architecture
  - Struts is one of the implementation of MVC arch.

+ **Struts2 MVC:**
  + View
    - Presentation Logic
    - Implemented as JSP pages
    - JSP pages will contain many struts tags e.g. <s:form/>
  + **Model**
    - Business Logic + Database Connectivity
    - Implemented as Java classes called as "Actions"
    - Typical action class contains
      - fields [to take input from prev page and to give output to next page]
      - param less constructor, getters/setters
      - Business Logic method (mostly execute())
    - Mostly this class is inherited from ActionSupport
  + **Controller**
    - Controls all navigations (view-model, model-view, view-view, model-model)
    - Implemented as Filter i.e. org.apache.struts2.dispatcher.FilterDispatcher
    - Information about all navigations is maintained in "struts.xml" in "src" directory.
    - Each user request first hit to the controller; then controller read struts.xml to find corresponding action class or jsp page and forward request to them.

+ Model(Action)-Model(Action) Navigation:
  Example: LoginAction -> SubjectAction

```
<action name="login" class="pkg.LoginAction">
    <result type="redirect-action" name="login_success">/subjects</result>
    <result name="login_fail">/error.jsp</result>
</action>
<action name="subjects" class="pkg.SubjectAction">
    ...
</action>
```

+ View-View Navigation:
  Example: error.jsp -> login.jsp
  in error.jsp

```
--------------
<s:a href="index.action">Login Again</s:a>
```

in struts.xml

```
--------------
<action name="index">
        <result>/login.jsp</result>
</action>
```

+ To access HttpSession, HttpServletRequest, HttpServletResponse or ServletContext into the action class:

- There are static methods into ServletActionContext class:

```
HttpServletRequest req = ServletActionContext.getRequest();
HttpServletResponse resp=ServletActionContext.getResponse();
ServletContext application=ServletActionContext.getServletContext();
HttpSession session = ServletActionContext.getRequest().getSession();
```

+ **.....Aware interfaces:**

- Another efficient way of accessing request, session or application attributes into the action is implementing RequestAware, SessionAware or ApplicationAware interface.
- Each of this inteface provide a setter method with a "Map" arg. This arg contains the attributes in form of java.util.Map from which they can get() or put().
- For Example, to access session attributes in the action class:

```
class MyAction extends ActionSupport implements SessionAware {
        // other fields, constructor, getters/setters

        private Map session;
        public void setSession(Map session) {
                this.session = session;
        }

        public void execute() {
                // ...
                // to add any value in session
                session.put("key1", value1);
                // ...
                // to retrieve any value from session
                value2 = (casting)session.get("key2");
        }
}
```

+ OGNL - Object Graph Navigation Language

- Used to access variables in different scopes (like EL)
- Used with struts tags - attribute values - in JSP pages
- Can access all values on Context Map:

+ context map
    |- application
    |- session
    |- value stack (root)
    |- request
    |- parameters
    |- attr

+ To access variables (attributes) from application, session or request scope use syntax:
    #application.varname, #session.varname, #request.varname

+ To access request parameters:
    #parameters.varname

+ To access variables (attributes) in any scope:
    #attr.varname
    This will search the varname into page scope to application scope.

+ All the values on value stack can be accessed without any marker '#'. The object of the current action is on the top of value stack (known as root) and hence all its data members can be accessed directly without using '#' marker.
    Example:
        <s:property value="outtext"/> <- out.jsp
            - outtext is field of CapAction.

+ #session.varname OR #session['varname']

+ Struts Tags:

```
<%-- <s:radio name="subject" list="subList"/> --%>
<s:iterator value="subList">
    <s:radio name="subject" list="{top}"/>
</s:iterator>

<%-- <s:checkboxlist name="book" list="bkList" listKey="bookid" listValue="name"/> --%>
<s:iterator value="bkList">
    <s:checkboklist name="book" list="{top}" listKey="bookid" listValue="name"/>
</s:iterator>
```

+ OGNL to EL:

- struts tags understand OGNL only (not EL).
- JSP tags, JSTL tags, Custom tags understand EL only (not OGNL).
- Sometimes there might be need of accessing OGNL values into JSP/Custom tags.
- To do this, we need to add those values into some scope using <s:set/> so that it can be accessed using EL syntax.
    <s:set name="varname" value="OGNL_expr" scope="page|request|session|application"/>

* Any struts tag cannot be nested (in attr value) into another struts/custom tag.

+ **Validation:**

- Action class can contain data validation logic (at server side).
- This validation can be performed into overridden validate() method of ActionSupport.
- In case any validation error, you can use addFieldError() or addActionError() method of the **ActionSupport.**
- The field level errors will be displayed on the input page nearby respective fields.
- The action errors will be displayed on the input page using <s:actionerror/> tag.
- In this case result with name "input" must be specified under that acion into struts.xml. For example:

```
<result name="input">/login.jsp</result>
```

- If any error is added into validate() method, execute() method will not be executed.
- Also some informative messages can be displayed on the JSP pages using <s:actionmessage/>. These messages must be added by the action using addActionMessage() of ActionSupport.

* Messages are NOT Errors.

+ **Action Business Logic:**

- By default execute() method contains business logic od the action.
- For sake of readability you may choose different name of the business logic method e.g. SubjectAction business logic may be written in a method with name fetchSubjects().

- In this case controller must be aware of the business logic method name, which can be done via struts.xml file:

```
<action name="subjects" class="pkg.SubjectAction" method="fetchSubjects">
    // ...
</action>
```

- login.jsp [user,pass]
  - LoginAction - checkLogin()
  - <action name="login" class="p.LoginAction" method="checkLogin">

- register.jsp[user,pass,cpass]
  - LoginAction - registerUser()
  - <action name="register" class="p.LoginAction" method="registerUser">

- chpass.jsp [user,pass,cpass]
  - LoginAction - changePassword()
  - <action name="chpass" class="p.LoginAction" method="changePassword">

+ **ModelDriven interface:**

- Typically Model is business object carrying data between action and view. It is a simple POJO class. Many times it represents data fetched from database (hibernate entity).

Prasanna Wadekar 9970248407

- Useful to avoid data redundancy.
- Example:

```
// Model class
class Book {
        // fields, constructor, getter/setters
}

// Action class
class AddBookAction implements ModelDriven<Book> {
        private Book book = new Book();
        public Book getModel() {
                return book;
        }
        public String execute() {
                // try-catch avoided for simplicity
                BookDao dao = new BookDao();
                dao.addBook(book);
                dao.close();
                return "success";
        }
}
```

- If action class is inherited from ModelDriven. controller invokes its getModel() method. Then data from html form is filled into the returned (from getModel()) model object. This object can be furthr used in business logic method.

+ Using Tiles with Struts 2:

1) create dynamic web project and add struts & tiles jar files into its WEB-INF/lib directory. Also add its "web.xml" file into that project.

2) create "user_master.jsp", which contains the basic html view that you want to apply for set of web pages. The configurable parts will be identified by some names like title, header, footer, body, etc using <tiles:insertAttribute> tag.

3) create default pages for above configurable parts i.e. header.jsp, footer.jsp, body.jsp, etc.

4) create the web pages, whose look and feel will be decided by base layout. i.e. subjects.jsp, books.jsp, showcart.jsp, etc.

5) create tile configuration file "tiles.xml" into WEB-INF directory.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
        "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
```

```
<tiles-definitions>
        // ...
</tiles-definitions>
```

This file contains basic html view config as :

```
<definition name="user_master" template="/user_master.jsp">
        <put-attribute name="title"  value=""/>
        <put-attribute name="header" value="/header.jsp"/>
        <put-attribute name="body"   value=""/>
        <put-attribute name="footer"   value="/footer.jsp"/>
</definition>
```

This file further contains one entry for each web page, which is extended from above "user_master" entry and override certain attributes like "title" and "body" as required. For Example:

```
<definition name="show_subjects" extends="user_master">
        <put-attribute name="title"  value="Subjects"/>
        <put-attribute name="body"   value="/subjects.jsp"/>
</definition>
```

6) Inside web.xml add context-param that informs about path of tiles.xml and also one TilesListener entry.

```
<context-param>
        <param-
name>org.apache.tiles.impl.BasicTilesContainer.DEFINITIONS_CONFIG</param-name>
        <param-value>/WEB-INF/tiles.xml</param-value>
</context-param>
<listener>
        <listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>
</listener>
```

7) finally go to struts.xml and declare a result type for "tiles".

```
<result-types>
        <result-type name="tiles" class="org.apache.struts2.views.tiles.TilesResult" />
</result-types>
```

8) In this file, modify the results for each web page, which will redirect you to the corresponding tile entry into "tiles.xml"

```
<action name="subjects" class="pkg.SubjectAction">
        <result type="tiles" name="show_sub">show_subjects</result>
</action>
```

+ Interceptors:

- Interceptors are java classes implementing "Interceptor" interface of struts2.
- Interceptor interface has 3 methods:
    - init()
        - immediately after creating object of interceptor
    - destroy()
        - before destroying object of interceptor
    - intercept()
    - code of this method can be splitted in 3 parts:
        - pre-processing: this part is executed before executing next element in the chain
        - invoke: invoke next element in chain and collect result
        - post-processing: this part is executed after executing next element in the chain
- **To write an interceptor:**
    step1: write an class inherited from Interceptor interface.
    step2: in intercept() method:
        - do pre-processing
        - invoke next ele in chain
        - do post-processing
    step3: make entry of the ineterceptor into struts.xml
    step4: add your interceptor into interceptor stack at proper place (create a modified interceptor stack) in struts.xml
    step5: apply interceptor stack to the intended action class in struts.xml

+ struts packages:

1. in struts.xml <package> is used to specify scope for global-results, default-interceptor-ref, etc.
    - So global-result and default-interceptor-ref are applicable only for the current package.

2. In package we can specify "namespace". This namespace is logical and will appear into the URL while accessing any "action".
    LoginAction
    |- <result name="sucess" type="redirect-action">customer/sub</result>

3. All JSP pages can be secured by putting them within WEB-INF; so that these pages are not directly accessible.
    - Further these pages can be stored under different folder so that maintaining web-site become easier. In that case all URLs should be modified accordingly.
    <result name="success">/WEB-INF/customer/subjects.jsp</result>

4. struts.xml can be divided into multiple xml files for sake of better maintenance.
    - In that case, XML files should be added in main struts.xml file.
    <include file="customer-struts.xml"/>

+ Struts Architecture:

https://struts.apache.org/docs/the-struts-2-request-flow.data/Struts2-Architecture.png

+ Spring Framework:

+ Introduction:

- Founder : Rod Johnson @ Year 2003
- Open Source Framework (Project) hosted on sourceforge.net
- Spring in Light Weight Comprehensive Framework for developing JavaSE and JavaEE application.
    - Light Weight
    - Comprehensive

+ Why Spring?

- Simplify Java Development
- Design with Interfaces
- Test Driven Application
- Declarative AOP (XML or Annotations)
- Easy coupling via Dependency Injection
    - Done by Spring IoC container [Inversion of Control]
    - Programmer declare fields and Spring Container initialize them via DI.
- Convert all checked exceptions into unchecked.
- Well designed - Easily Extendable - Reusable
- Extremely Modular - Flexible - Easily integrable with any existing technology

+ Spring Architecture:

http://docs.spring.io/spring/docs/current/spring-framework-reference/html/images/spring-overview.png

+ Java SE Application With Spring:

- step1: create new "Java Project".
- step2: create a "lib" folder in the project and copy required spring jars into it.
- step3: add all the jars into the classpath of the project.

- step4: in "src" directory create "Spring Bean Configuration File" with any name e.g. spring1.xml
- step5: create interface IPerson with getter/setter declarations.
- step6: implement interface into class Person
- step7: add "Person" entry into xml file and set some props

```
<bean name="p1" class="sun.krd.spring.basic01.Person">
        <property name="name" value="Nilesh"/>
        <property name="address" value="Pune"/>
        <property name="age" value="28"/>
</bean>
```

- step8: write a main class, in which create "XmlBeanFactory" object and retrieve "Person" object through its "getBean()" method. Finally call the "Person" methods from interface reference.

```
Resource res = new ClassPathResource("spring1.xml");
XmlBeanFactory factory = new XmlBeanFactory(res);

IPerson p = (IPerson) factory.getBean("p1");
System.out.println("Name : " + p.getName());
System.out.println("Addr : " + p.getAddress());
System.out.println("Age : " + p.getAge());
```

+ **Dependency Injection:**

- In above example, the object of Person class is created by the spring container (during call to getBean()).
- During creation by default param less constructor is called, and then container call the setter methods as given by <property ... /> tags in xml.
- This way of injecting values of data members is referred as "setter based" DI.

- If needed, values of data members can be passed via parameterized constructor.
- To do this spring xml configuration should be done as :

```
<bean name="p2" class="sun.krd.spring.basic01.Person">
    <constructor-arg index="0" value="Raj" />
    <constructor-arg index="1" value="Kaard" />
    <constructor-arg index="2" value="24" />
</bean>
```

- This way of injecting values of data members is referred as "constructor based" DI.

+ init-method and destroy-method:

- It is also possible to call some method after creation of the object (after constructor execution and DI) by specifying method name into xml file.
- Also it is possible to call some method before destruction of the object.
- These method typically contains additional initialization and de-initialization logic.
- Example:

```
class Person {
    // ...
    public void initPerson() {
        // ...
    }
    // ...
}

<bean name="p3" class="pkg.Person" init-method="initPerson">
    // ...
</bean>
```

* Alternatively (instead of declaring in xml file), these methods may have @PostConstruct and @PreDestroy annotations.

+ **XmlBeanFactory:**
- When XmlBeanFactory is created, it loads xml file and check its syntax.
- It is used to get the bean reference (by name or by type).
- When getBean() is called, factory checks whether bean object is already created. If object is already created, it simply returns reference to that object.
- If object is not created, factory will load the given class, create the object, performs DI and call init method if given.

+ **ApplicationContext:**
- When ApplicationContext is created, it will load and validate spring xml file.
- Also during creation, it will create all bean objects, performs DI and call init method if given.
- When getBean() method is called, reference of the bean will be returned.
- ApplicationContext provides all functionalities of XmlBeanFactory and also provides few others:
    - event handling & publishing
    - internatialization
    - resource management and access
    - life cycle interfaces
- two types:
    - ClassPathXmlApplicationContext
        - used with xml config of spring
        - To create context object

```
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("spring1.xml");
ctx.registerShutdownHook();
```

        - call to registerShutdownHook() ensures that all bean objects will be destroyed when spring container will be closed.

    - AnnotationConfigApplicationContext
        - used with annotation config of spring
        - to perform annotation config (no xml) write a seperate class (other than interface and bean class)

```
@Configuration
public class SpringBeanConfig {
    @Bean
    public Object p1() {
        Person p = new Person();
        p.setName("ABC");
        p.setAddress("XYZ");
```

```
                p.setAge(123);
                return p;
            }
        }
        - in main()
            AnnotationConfigApplicationContext ctx = new
AnnotationConfigApplicationContext(SpringBeanConfig.class);
            ctx.registerShutdownHook();

            IPerson p = (IPerson) ctx.getBean("p1");
```

+ Dependency and Dependent Beans:

1. interface ILog
   - void write(String str);
2. class ConsoleLog implements ILog
3. class FileLog implements ILog
4. interface IPerson
   - getters/setters for name, address, age and logger.
5. class Person implements IPerson
   - Fields: name, address, age
     private ILog logger;
   - implement all methods of IPerson
     - in getter/setter of name, address and age
       - logger.write("method() called");
6. spring xml file

```
<!-- dependency declaraions --!>
    <bean name="consLog" class="pkg.ConsoleLog">
    </bean>
    ... FileLog

<!-- dependent declaraions --!>
    <bean name="p1" class="pkg.Person">
        <setproperty name="logger" ref="consLog"/>
        .... other setters
    </bean>
```

7. main()
   use ClassPathXmlApplicationContext to get Person bean and display its details.

+ Spring Config:
  1. XML config - Only XML (no anns)
  2. Annotation config - Only Anns (no XML)
  3. Java Config - XML + Annotations

+ **Spring XML file:**
- beans (3.0), context (3.0)
- <context:annotation-config/>
- Spring container will recognize annotations like @PostConstruct, @PreDestroy, @Autowired, @Qualifier, etc. on all beans.

+ **@Autowired**
- Used on the fields of the bean, so that container will perform DI for those fields.
- When @Autowired is used on any field, by default container will search all the beans of matching type (same type or derived type) and if single bean is found then it is injected (attached to the field).
- If more than one matching bean is found, then autowiring fail and exception is raised.

+ **@Qualifier**
- Mainly used to resolve the ambiguity raised during autowiring, by specifying name of the dependency bean.
-

```
@Autowired
@Qualifier("consLog")
private ILog logger;
```

+ stereo-type annotations:
- These annotations are used on class level, so that those classes will be treated as spring beans by the container.
- To search all the classes with stereo type anns in certain package use

```
<context:component-scan base-package="pkgName">
</context:component-scan>
```

- there are four stereo type annotations:

1. @Component
- All other bean classes like simple POJOs should be annotated with @Component.

2. @Service
- Bean classes containing business logics should be annotated with @Service. These classes are typically used business logic layer. Many times these classes internally access repositories.

3. @Repository
- Bean classes doing data handling, database connections (in which data can be read or written) should be annotated with @Repository. These classes are typically used in backend (persistence) layer.

4. @Controller
- Bean classes performing navigation, model-view binding should be annotated with @Controller.

+ **Spring Bean scopes:**
1. singleton:

- default scope
- when application context is loaded, one bean object is created
- each call to getBean() returns same object

2. prototype:
   - when application ctx is loaded, no bean object is created.
   - each call to getBean(Class cls) returns a new object.
3. request:
   - scope is limited to current request
   - used only in web application
4. session:
   - scope is limited to current session
   - used only in web application

* Spring bean scopes can be declared into spring xml file <bean /> tag or @Scope anns on the bean class.

```
@Component
@Scope("prototype")
class Book {
	// ...
}
```

+ Using Stereotype annotations for implementing simple assign:

```
- @Component
@Scope("prototype")
class Book {
	// ...
}
- @Repository("bkDao")
class BookDao {
	private List<Book> bkList;
	// ctor, getter/setter
	// insert(), delete(), search(), ...
}
- @Service
class BookService {
	@Autowired
	private BookDao dao;
	// ctor, getter/setter
	// addBook(), delBook(), findBook(), getBookPrice() ...
}
- class BookMain {
	main() {
		// menu driven program
```

```
                // BookService obj -> ctx.getBean(BookService.class);
                // Book obj -> ctx.getBean(Book.class);
        }
    }
    - spring xml file
    <context:component-scan base-package="sun.krd.spring.shop"/>
    <bean name="bkDao" class="sun.krd.spring.shop.BookDao">
        <property name="bkList">
            <list>
                <bean class="sun.krd.spring.shop.Book">
                    <property name="name" value="book1"/>
                    <property name="price" value="11"/>
                </bean>
                ...
            </list>
        </property>
    </bean>

+ SPEL : Spring Expression Language
    - Can be used in JSP pages and also in spring bean classes using @Value annotation.
    - syntax: #{var.member}, #{var.method()}
    - in bean classes @Value annotation is used for DI (instead of @Autowired and @Qualifier)
    Example:
        class BookService {
            @Value("#{bkDao}")
            private BookDao dao;
            // ...
        }
+ Assign:
    class AccHolder {
        String name, address, mobile;
        // ...
    }
    class Account {
        int accid;
        String acctype;
        double balance;
        * use SPEL
        AccHolder accHolder;
        // ...
        // withdraw(), deposit()
    }
    class AccountMain {
```

```
        main() {
                // create Account Object
                // get its data from user
                // perform withdraw(), deposit()
                // print balance after each operation
        }
}
```

+ **Aspect Oriented Programming:**
  - The business logic is seperated from cross-cutting concerns
  - The cross-cutting concerns are different aspects other than actual business logic e.g. logging, monitoring, security, etc
  - Due to this code will be modularized and business logic, other concerns will be implemented seperately (by different developers).

+ AOP Terminologies:
  - Aspect:
    - Additional fuctionality (cross cutting concern) other than business logic
    - In spring, implemented as a seperate class annotated with @Aspect
  - Advice:
    - The operation to be performed to handle certain cross-cutting concern
    - In spring it is implemented as methods in aspect class
    - There are four types of advices given by different annotations (on advice methods):
      - @Before - pre-processing
      - @After - post-processing
      - @Around - pre-processing and post-processing
      - @AfterThrowing - post-processing in case business logic method throws exception
  - JoinPoint:
    - represent the point in the application where advice is to be applied.
    - In spring it is an "JoinPoint" object which contains information of the business logic on which advice is applied. It also contains information about the object on which business logic method is called.
  - **Target:**
    - The object on which business logic method is called.
  - Pointcut:
    - combination of one or more join points where advice is to be applied
    - In spring PointCut is implemented as an empty method with @PointCut annotation
  - Proxy:
    - This object is internally created by the spring container to call the advices and the actual business logic.
  - Advisor:
    - group of advice and pointcut into a single unit called as "Advisor".
    - This object is internally passed to proxy factory to create the proxy object.

+ AOP Example:

- create new java project
- create a "lib" folder and add spring jars as well as aop jars.
- add these jars into the classpath of the project.

- implement interfaces, beans in the project
    - interface IAccount
    - class Account
        - accid, type, balance
        - constructors/getter/setters
        - deposit()/withdraw()
- create spring xml file with bean(3.0), context(3.0), aop(3.0).
- into spring xml file add usual tags <context:component-scan ... />

- @Component

```
@Aspect
class AccountAspects {
	@Before("execution (* pkg.Account.set*(..))")
	public void logBefore(JoinPoint pt) {
		System.out.println("BEFORE : " + pt.getSignature());
	}
	@After("execution (* pkg.Account.get*(..))")
	public void logBefore(JoinPoint pt) {
		System.out.println("AFTER : " + pt.getSignature());
	}
	@Around("execution (* pkg.Account.withdraw(..))")
	public Object monitorTransaction(ProceedingJoinPoint pt pt) throws Throwable {
		IAccount acc = (IAccount)pt.getTarget();

		System.out.println("Balance Before : " + acc.getBalance());

		Object res = pt.proceed();

		System.out.println("Balance After : " + acc.getBalance());

		return res;
	}
}
```

- in spring xml file:

```
	<aop:aspectj-autoproxy/>
```

- Spring implementation of AOP is based on open source AOP project "AspectJ".

+ **Spring Web MVC Architecture:**
- Used for implementing MVC into web applications.
+ MVC components are as follows:
+ View:
- implemented as JSP pages
- The JSPs can contain standard tags, JSTL tags, Velocity or any other third tag libraries.
- Also provide support for SPEL and JSP EL.
- Internally views are resolved by "ViewResolver" class.
+ **Controller:**
- The front controller is implemented by the spring mvc in form of servlet "DispatcherServlet" (need to make entry into web.xml).
- This controller is responsible for navigation control (flow).
- The controller is associated with a xml file in WEB-INF folder named as <spring-servlet-name>-servlet.xml.

web.xml

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/*</url-pattern>
</servlet-mapping>
```

- In this case, spring xml configuration should be created in WEB-INF with name spring-servlet.xml.
- For simplest application this file may have beans(3.0), context(3.0), mvc(3.0)

- The programmer can create one or more conroller classes.
- These controller classes are used to bind Model and View.
- Also these classes will contain one more request handlers (methods) which in turn execute the business logic typically from the service class (@Service).
- The request handlers will be resolved using "HandlerMapping" object.

+ **Model:**
- In spring Model is simple POJO classes.
- They are used to transfer data between View and Controller.

+ **Spring Web MVC Architecture (Flow)**
http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/images/mvc.png

+ **Steps for Spring Hello Web MVC:**

- create new "dynamic web project" e.g. spring_mvc
- in "WEB-INF/lib" folder add all spring jars (basic, mvc, tiles, hibernate, aop, etc).
- in web.xml make entry for spring front controller - DispatcherServlet
- in WEB-INF create spring config file (spring-servlet.xml) with beans(3.0), context(3.0) and mvc(3.0)
- in spring xml file:

```
<context:component-scan .../>
<mvc:annotation-driven/>
<bean id="viewResolver"
class="org.springframework.web.servlet.view.UrlBasedViewResolver">
        <property name="viewClass"
        value="org.springframework.web.servlet.view.JstlView"/>
        <property name="prefix" value="/" />
        <property name="suffix" value=".jsp"/>
</bean>
```

****

- in WebContent, create "first.jsp"

```
<h2>First Page</h2>
<a href="/spring_mvc/link_second">Second Page</a>
```

- in WebContent, create "second.jsp"

```
<h2>Second Page</h2>
```

- in src,

```
@Controller
class HelloController {
        @RequestMapping("/link_second")
        public String handleSecondLinkRequest() {
                return "second";        // name of next jsp
        }
}
```

****

- In WebContent/pages, create "third.jsp"

```
<h2>Third Page</h2> <hr/>
<a href="/spring_mvc/link_fourth">Next Page</a>
```

- In WebContent/pages, create "fourth.jsp"

```
<h2>Fourth Page</h2> <hr/>
Welcome Again!! <br/>
Today : ${today}
Book Details : ${bk.bookid}, ${bk.name}, ${bk.price}
```

- in HelloController class:

```
@RequestMapping("/link_fourth")
public String handleFourLinkRequest(Model model) {
        String curDate = new Date().toString();
```

```
        model.addAttribute("today", curDate);
        Book b = new Book(101, "book name", 123.50);
        model.addAttribute("bk", b);
        return "pages/fourth";
    }
```

****

1. in WebContent/caps -> index.jsp

```
        <a href="/spring_mvc/input_link">Input Page</a>
```

2. CapModel class: (to add as model attribute)
   - private String inText;
   - constructor, getter, setter
3. CapsController class:

```
    @RequestMapping("/input_link")
    public String gotoInputPage(Model model) {
        CapModel cm = new CapModel();
        model.addAttribute("cm", cm);
        return "caps/input";
    }
```

4. in WebContent/caps -> input.jsp

```
    <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

    <form:form modelAttribute="cm" method="post" action="/spring_mvc/output_link">
        Enter Text : <form:input path="inText"/>
        <input type="submit" value="Submit"/>
    </form:form>
```

5. CapsController class:

```
    @RequestMapping("/output_link")
    public String gotoOutputPage(Model model, @ModelAttribute("cm") CapModel cm) {
        String text = cm.getInText();
        text = text.toUpperCase();
        model.addAttribute("outputText", text);
        return "caps/output";
    }
```

6. in WebContent/caps -> output.jsp

```
    Input Text : ${cm.inText}
    Caps Text : ${outputText}
```

+ Request handler method:

- The request handler methods (annotated with @RequestMapping) can have very flexible signature.
- It can take arguments of different types in any order:
  - Model / ModelMap / Map - data carried from controller to view or vice versa.
  - argument with @ModelAttribute - for easy access of model data.

- HttpSession - session object
- HttpServletRequest, HttpServletResponse
- argument with @CookieValue("cookieName") - for easy access of cookie data
- BindingResult - to add validation errors (usually written as last argument)

- It can also have different return types:
  - String - return view (jsp) name
  - ModelAndView - return ModelAndView object containing view name, model attr name, model attr object.
  - View - return View object containing jsp name

  * For redirection from req handler method to another req handler method use "redirect:reqUrl" as view name.

```
@RequestMapping("/one")
String handleOne() {
	// ...
	return "nextjsp";
}
@RequestMapping("/two")
String handleTwo() {
	// ...
	return "redirect:one";
}
```

+ **Spring 3 - Tiles 2 Steps:**

1. In the spring project, ensure that tiles jars are present. tiles-api-2.2.2.jar, tiles-core-2.2.2.jar, tiles-jsp-2.2.2.jar, tiles-servlet-2.2.2.jar, tiles-template-2.2.2.jar, commons-digester-2.0.jar, commons-beanutils-1.8.0.jar

2. Configure TilesView url based resolver and tiles configurer in the spring config file.

```
<bean id="tilesConfigurer"
	class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
	<property name="definitions">
		<list>
			<value>/WEB-INF/tiles.xml</value>
		</list>
	</property>
</bean>

<bean id="viewResolver"
	class="org.springframework.web.servlet.view.UrlBasedViewResolver">
	<property name="viewClass"
		value="org.springframework.web.servlet.view.tiles2.TilesView" />
</bean>
```

3. Create a folder under WEB-INF and create master pages (templates) as well as header, footer and menu pages in it.

Note that master pages have <tiles:insertAttribute/> tags in which "name" attribute represent some name (linked with tiles.xml).

4. Create the content pages in the same folder or seperate folder. Note that content pages contains simple spring GUI stuff. No need to repeat <html> or <body> tag again as they are already included into master pages.

5. Create tiles.xml under WEB-INF folder. This file is linkage between master page and content pages.

```
<definition name="headerfooter.definition" template="/WEB-INF/jsp/headerfooter_master.jsp">
    <put-attribute name="title" value="" />
    <put-attribute name="header" value="/WEB-INF/jsp/header.jsp" />
    <put-attribute name="body" value="" />
    <put-attribute name="footer" value="/WEB-INF/jsp/footer.jsp" />
</definition>
```

The above tag defined master page in tiles.xml. Note that master page definition has got a "name" which will be used to define content pages. Also <put-attribute> "name" is mapped to <tiles:insertAttribute> "name" in the master pages.

```
<definition name="login" extends="headerfooter.definition">
    <put-attribute name="title" value="Login" />
    <put-attribute name="body" value="/WEB-INF/pages/login.jsp" />
</definition>
```

The above tag defined content page in tiles.xml. Note that content page "extends" from master page and <put-attribute> override the values given in master page definition. Finally "name" in the definition of content page will identify the "view" from the controller.

* String (ViewName) returned from controller will represent tile-defn in tiles.xml

+ Spring 3 - Hibernate 3 Steps:

1. In the spring project, ensure that hibernate jars are present.
2. configure hibernate into hibernate.cfg file - dialect, show_sql, entity classes
3. configure datasource in spring xml file

```
<bean id="dataSrc" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@pdc1:1521:oracle" />
    <property name="username" value="smita" />
    <property name="password" value="smita" />
    <property name="initialSize" value="1" />
```

```
                <property name="maxActive" value="2" />
        </bean>
```

4. configure hibernate in spring xml file

```
        <!-- hibernate configuration -->
        <bean id="hbSF"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
                <property name="dataSource" ref="dataSrc"/>
                <property name="configLocation" value="classpath:hibernate.cfg.xml"/>
        </bean>

        <!-- hibernate transaction configuration -->
        <bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
                <property name="sessionFactory" ref="hbSF" />
        </bean>

        <tx:annotation-driven transaction-manager="txManager"/>
```

5. in dao class, autowire hibernate SessionFactory "hbSF" and access current hibernate session to perform hibernate operations. Note that, the dao class or its methods should be marked as @Transactional.

+ Spring 3 Security Steps:

1. In the spring project, ensure that tiles jars are present.

spring-security-acl-3.1.0.RELEASE.jar, spring-security-config-3.1.0.RELEASE.jar, spring-security-core-3.1.0.RELEASE.jar, spring-security-crypto-3.1.0.RELEASE.jar, spring-security-taglibs-3.1.0.RELEASE.jar, spring-security-web-3.1.0.RELEASE.jar

2. In web.xml, configure spring application context

```
        <!-- Spring Application Context -->
        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
        </listener>
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>
                        /WEB-INF/spring-servlet.xml
                </param-value>
        </context-param>
```

3. Also, in web.xml configure spring security filter

```
        <filter>
                <filter-name>springSecurityFilterChain</filter-name>
```

```
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

4. Create LOGIN database :

CREATE TABLE spring_users (USER_ID INT(10) PRIMARY KEY, USERNAME VARCHAR(45), PASSWORD VARCHAR(45), ENABLED tinyint(1));

CREATE TABLE spring_roles (USER_ROLE_ID INT(10) PRIMARY KEY, USER_ID INT(10) REFERENCES spring_users(USER_ID), AUTHORITY VARCHAR(45));

5. In spring conf file, configure spring security

```
    <security:http auto-config="true">
        <security:intercept-url pattern="/book/*" access="ROLE_USER" />
        <security:form-login login-page="/login" default-target-url="/book/newcart"
            authentication-failure-url="/login" />
        <security:logout logout-success-url="/logout" />
    </security:http>

    <security:authentication-manager>
      <security:authentication-provider>
        <security:jdbc-user-service data-source-ref="dataSrc"
          users-by-username-query="select username,password,enabled from spring_users
        where username=?"
          authorities-by-username-query="select u.username, ur.authority from spring_users u,
        spring_roles ur where u.user_id=ur.user_id and u.username=?"/>
      </security:authentication-provider>
    </security:authentication-manager>
```

6. Create login.jsp with action "j_spring_security_check" and html textboxes as "j_username" and "j_password".

+ Java Server Faces:

- JavaServer Faces (JSF) is a Java specification(std part of J2EE specs , J2EE 1.5 onwards) for building component-based user interfaces for web applications.
- It's a standard Java EE web mvc framework.
- Well-designed and easy-to-use component-based web framework.

- JSF is based on well established MVC(Front Servlet Controller) design pattern.
- There are as many popular JSF implementations, namely Oracle Mojarra and Apache MyFaces & PrimeFaces, RichFaces etc.
- It helps to create web applications with server-side user interfaces (UIs).

+ JavaServer Faces technology consists of the following:

1. An API for representing components and managing their state; handling events, server-side,validation, and data conversion; defining page navigation; supporting internationalization etc.
2. Tag libraries for adding components to web pages and for connecting components to server-side objects.

+ A typical JavaServer Faces application includes the following parts:

1. Web pages in which components are laid out.
2. JSF custom tags to add components to the web page
3. Managed beans, which are lightweight container-managed objects (POJOs) based upon dependency injection, lifecycle callbacks and interceptors.
4. Web deployment descriptor (web.xml file)
5. Optional, configuration files -- faces-config.xml file, to define page navigation rules and configure beans and custom components.
6. Custom objects --- custom components, validators , converters, or listeners --programmer created.
7 A set of custom tags for representing custom objects on the page.

+ With minimal effort, you can complete the following tasks.

1. Create a web page.
2. Drop components onto a web page by adding component tags.(from JSF tag libraries)
3. Bind components on a page to server-side data.
4. Wire component-generated events to server-side application code.
5. Save and restore application state beyond the life of server requests.
6. Reuse and extend components through customization.

+ Developing a simple JavaServer Faces application typically requires the following steps:

1. Create JSF enabled web application.
Dynamic web project -- Choose configuration -- JSF 2.1 , disable libraries (since jsf libs will be added later to <lib>), configure Faces Servlet , additional url patterns can be added -- *.jsf, *.xhtml.

2. Change compiler compliance level to 1.7 & change from project facet -- Java 1.7

3. Copy jsf jars under <WEB-INF>/lib.

4. Developing managed beans(equivalent to Controller in Spring , actions in Struts or Java beans in Standalone MVC) eg : HelloBean Annotations-

```
@ManagedBean(name="nameOfBean/id", eager="def:false")
default scope --- request scope.

@ManagedBean(name="hello")
public class HelloBean {
        private String message;
        // constructor, getter/setters
}
```

5. Create web pages using component tags

```
<html lang="en"
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html">
      <h:head>
            <title>Facelets Hello </title>
      </h:head>
      <h:body>
            #{hello.message}
      </h:body>
</html>
```

6. Mapping the javax.faces.webapp.FacesServlet instance -- Is already done at the time of creating JSF enabled web application.

+ Some more JSF annotations.

1. Scope related Annotations
   - @RequestScoped-- def scope --current request only.
   - @ViewScoped -- Bean lives as long as user is interacting with the same JSF view in the browser window or tab. It gets created upon a HTTP request and get destroyed once user postbacks to a different view.
   - @SessionScoped -- current session
   - @ApplicationScoped -- entire application
   - @NoneScoped -- no scope
   - Custom scope can also be created.

2. Dependency Injection annotation -- to inject some other bean .

   Eg -- In managed bean -- UserBean

```
@ManagedBean(name="ub")
public class UserBean {
        @ManagedProperty(value="#{adr}")
```

```
        private Address address;
        // ...
}
-- in another managed Bean address
@ManagedBean(name="adr")
public class Address {
        // .....
}
```

* If eager=true & scope=application, only then bean is instantiated at the application start up time.

+ JSF life cycle.

- It involves --- the client makes a request for the web page, and the server responds with the page.
- The lifecycle consists of two main phases: execute and render.

During the execute phase:

1. The application view is built or restored.
2. The request parameter values are applied.
3. Conversions and validations are performed for component values.
4. Managed beans are updated with component values.
5. Application logic is invoked.

* For a first (initial) request, only the view is built. For subsequent (postback) requests, some or all of the other actions can take place.

During the render phase:

1. the requested view is rendered as a response to the client. Rendering is typically the process of generating output, such as HTML or XHTML, that can be read by the client, usually a browser.

+ JSF Internals:

Any JSF application goes through the following stages when it is deployed on the server.

1. When the application is built and deployed on the the application is in an uninitiated state.
2. When a client makes an initial request for the hello.xhtml web page, the hello Facelets application is compiled.
3. The compiled Facelets application is executed, and a new component tree is constructed for the hello application and is placed in a javax.faces.context.FacesContext.
4. The component tree is populated with the component and the managed bean property associated with it, represented by the EL expression hello.message
5. A new view is built, based on the component tree.
6. The view is rendered to the requesting client as a response.
7. The component tree is destroyed automatically.
8. On subsequent (postback) requests, the component tree is rebuilt, and the saved state is applied.

+ JSF Advantages:

1. JSF is based on well established Model-View-Controller (MVC) design pattern.
2. Applications developed using JSF frameworks are well designed and easy to maintain than any other applications developed in JSP and Servlets.
3. JSF provides standard, reusable components for creating user interfaces for web applications.
4. JSF provides many tag libraries for accessing and manipulating the components.
5. JSF is a specification and vendors can develop the implementations for JSF.

+ Enterprise Java Beans

- Enterprise applications contains number of presentation elements, a lot of business logic (for different purposes) and one or more databases.
- Most of the times there is need of executing business logic from different presentation elements (like web applications, desktop applications or mobile applications). EJBs are used to serve this purpose.

- What is EJB?

EJB is a java class containing business logic, which runs on an application server as per user request. EJB objects are created and managed by the EJB container.

- Why use EJBs?(Benefits)

1. EJBs simplify the development of large, distributed applications -scalable,reliable,secure
2. EJB container provides system-level services(Primary services JNDI, conn. pooling, Transactions, security:authorization, authentication, persistence: JPA) to the bean developer can concentrate on solving business problems.
3. For thick clnts : as the EJBs rather than the clients contain the application's business logic, the client developer can focus on the presentation(view) of the client. As a result, the clients are thinner, especially suitable for clients that run on small devices.(embedded)
4. EJBs are portable & reusable components. The application assembler can build new enterprise applications from existing beans.-- compliant on any Java EE server(application server)

- There are different types of EJBs:

1. session beans :

- bean objects are always accessed via interfaces. Any type of session bean can support three types of interfaces:

1. @Remote: this interface is used to access bean methods from outside the ejb container.
2. @Local: this interface is used to access bean methods from within the ejb container (from some other ejb).
3. @EndPoint: this interface is used only for stateless beans to implement web services.

+ stateless bean:

Once an instance is in the Method-Ready Pool, it is ready to service client requests. When a client invokes a business method on an EJB object, the method call is delegated to any available instance in the Method-Ready Pool. While the instance is executing the request, it is unavailable for use by other EJB

objects. Once the instance has finished, it is immediately available to any EJB object that needs it. Stateless session instances are dedicated to an EJB object only for the duration of a single method call.

Life Cycle:

1. when request is made of stateless bean, if it "does not exist", ejb container creates object of the bean.
2. then it calls @PostConstruct method, if written. We can write initialization code here.
3. Now bean object go to "ready" state. Any business logic method can be executed only if bean is in ready state.
4. At the end of life cyle, ejb container calls @PreDestroy method. We can write de-initialization code here.
5. Then bean object go to "does not exist" state. Now it is ready for garbage collection.

```
@Remote
public interface MathIntf {
        int add(int a, int b);
}

@Stateless(mappedName="ejb/math")
public class MathImpl implments MathIntf {
        public int add(int a, int b) {
                return a + b;
        }
}

// client side
InitialContext ctx = new InitialContext();
// initialize ctx, if accessing from outside the application server
MathIntf obj = (MathIntf)ctx.lookup("ejb/math");
int res = obj.add(10, 20);
```

+ stateful bean:

Each SFSB is dedicated to one client for the life of the bean instance; it acts on behalf of that client as its agent. Stateful session beans are not swapped among EJB objects, nor are they kept in an instance pool like their stateless session counterparts. Once a stateful session bean is instantiated and assigned to an EJB object, it is dedicated to that EJB object for its entire lifecycle.

Life cycle:

1. when a request come from new client, new object of the bean in created by the EJB container.
2. then EJB container calls @PostConstruct method, if written.
3. now bean object is in ready state. Client can call any business logic methods.
4. In low memory situations, ejb container may store the bean object in some temporary storage. In this stage, bean object cannot be used. So this stage is called as "Passive" state. Mostly ejb containers use LRU algorithm to decide which bean to be passivated.
5. EJB container calls @PrePassivate method before passivating bean object, if written.

Prasanna wadekar

6. If client requests for passivated bean, ejb container bring the bean object into the memory and make it "ready" for user requested operation.

7. before activation, ejb container calls @PostActivate method, if written.

8. When user want to release the bean object (when no longer needed), he calls @Remove method, if written.

9. After this ejb container calls @PreDestroy method, if written.

10. Now the bean object is eligible for garbage collection.

11. If bean is not used for very long time, ejb container can destroy the bean.

```
@Remote
public interface MathIntf {
	void setNum1(int a);
	void setNum2(int b);
	int add();
}

@Stateful(mappedName="ejb/math")
public class MathImpl implments MathIntf {
	private int a, b;
	public void setNum1(int a) {
		this.a = a;
	}
	public void setNum2(int b) {
		this.b = b;
	}
	public int add() {
		return a + b;
	}
}

// client side
InitialContext ctx = new InitialContext();
// initialize ctx, if accessing from outside the application server
MathIntf obj = (MathIntf)ctx.lookup("ejb/math");
obj.setNum1(10);
obj.setNum2(20);
int res = obj.add();
```

2. message driven beans

- Message Driven Beans are used when we need one way communication (from client to MDB).
- Here client simple send data to MOM and continue its own work.
- MOM transfers that data to the interested/corresponding MDB.

+ The messaging service can have one of the following model:

1. Queue : Point To Point Model
   - Durable Message
   - Msg will be sent if MDB is available, If not available, it will be pending in Queue
2. Topic : Publisher Subscriber Model
   - Non-Durable Message
   - Msg will be sent if MDB is available; If not available, msg will be lost.

- MDB is Enterprise Java Bean (Java Class) which is interested in receiving message from MOM.

```
@MessageDriven
// ...
public class SampleMDB implements MessageListener {
	public void onMessage(Message message) {
		System.out.println("Message Received!");
	}
}
```

+ Life cycle of MDB:

1. After creating the object of mdb, ejb container calls @PostConstruct method, if written.
2. When MDB is ready, it can receive messages from MOM i.e. its onMessage() will be called.
3. At the end of life cycle, ejb container will call @PreDestroy method, if written.
4. Then MDB object is eligible for garbage collection.

* Note, most of the times ejb containers create a pool of mdb objects and when message arrives, one of the object's onMessage() method is called.

3. entity beans:

+ Till ejb 2.1, there were three of ejbs:
  - session beans, message driven, entity beans (db)
+ In ejb 3.0 specification, there are only two types of ejbs:
  - session beans, message driven
+ Instead of entity beans a new specification of db connectivity is added known as JPA 1.0. JPA is an ORM specification.

+ Java Persistence API:

- JPA is specification given by SunMicrosystem for ORM.
- It is just like hibernate (as learnt before), with main difference being it is a specification; while hibernate is a product. Hibernate is one of the implementor of this specification.

- JPA entity life cycle:

1. **New:** When an entity object is initially created its state is New. In this state the object is not yet associated with an EntityManager and has no representation in the database.
2. **Managed:** An entity object becomes Managed when it is persisted to the database via an EntityManager's persist method, which must be invoked within an active transaction.
3. **Detached:** A managed entity object can also be retrieved from the database and marked for deletion, by using the EntityManager's remove method within an active transaction. The

entity object changes its state from Managed to Removed, and is physically deleted from the database during commit.

4. **Removed:** The last state, Detached, represents entity objects that have been disconnected from the EntityManager. For instance, all the managed objects of an EntityManager become detached when the EntityManager is closed.

- **JPQL is a query language much similar to HQL.**

|| श्री गणेशाय नमः ||

SUNBEAM

Institute of Information Technology

Placement Initiative

C Language

1. Explain flow of compilation and execution of C program?

main.c → Preprocessor → main.i → compiler → main.Asm → Assembler → main.obj → .lib (library file)

main.c - Source code main.i - intermediate file

2. List name of five keywords given by ANSI?

enum, signed, const, volatile, void

3. Write a program to find out maximum of three numbers using conditional operator?

```
int a,b,c;
(a>b) && (a>c) ? a : (b>c) ? b : c
```

4. Why array index always starts with zero?

Normally, compiler converts arr notation into pointers e.g. arr[1] = *(arr[0]+1) = 10 i.e. location 1 in array, so we are skipping elements. So to access it,

5. How will you find out largest element without temporary variable in array?

```
for(i=0;i<n;i++)
{ for(j=1;j<n;j++)
  { if a[i]<a[j]
    { a[i]=a[i]+a[j] //swap
      a[j]=a[i]-a[j]
```

```
a[i]=a[i]-a[j]
pf("%d", a[0]);
```

6. What is the significance of storage class? List name of storage classes in C?

When we consider any variable we should consider its 2 parameters which are 1) life and 2) scope. Life means existence of variable in memory and scope means its accessibility. 4 issues we need to consider about variables are 1) storage, 2) life, 3) scope, 4) default value

PRASANNA

macros can't be

C Language

7. What do you know about declaration and definition of variable?

Declaration of variable means we announce to compiler which variable we want to use in future and definition of variable means memory gets allocated to variable.

8. What is recursion and what is prerequisite for it?

In recursion we think of defining process in terms of itself and about terminating condition. These are prerequisite of recursion. recursion gives simplicity in code.

9. What is conditional compilation?

Conditional compilation is instruction given to compiler that compile specific code only if condition in which it is written is true otherwise don't. e.g. #if #else conditional compilation is normally used in device driver coding

10. What is the difference between function and macro?

fun are called at compile time and macros are processed at preprocessing level. fun are type safe and macro are not type safe. fun is slower and macro is fast. when we want to do big code use fun & for small code use macro. fun can be recursive & *

11. What is the significance of function calling convention? List name of function calling conventions? Which is default function calling convention?

Calling convention deals with argument pushing issues such as which fun will pop arguments (either caller fun or called fun). Argument popping responsibility is also called as stack cleanup. calling conventions are -

12. What is function activation record? What it contains?

function activation record is created on stack when fun is called. It contains local variables, formal arguments and return address (address of next instruction). when function is completed FAR is destroyed.

C Language

13. What is the difference between #include <xyz> and #include "xyz"

#include<xyz> this statement searches specified header file in standard library and #include "xyz" this will search specified header file in current project

14. What is the difference between typedef and #define?

typedef is keyword used to give another name to existing datatype typedef is handled by compiler e.g. size_t is used for unsigned int. #define is preprocessor directive which directs preprocessor to replace value of 14

15. What is pointer? Which arithmetic operations we can perform on pointer?

pointer is one variable which holds address of memory location. Internally it is unsigned integer and never negative. pointer size is 4 bytes in visual studio. we can increment or decrement pointers. we can also subtract 2

16. Can you explain what is dangling pointer and memory leakage? How can we detect memory leakage?

dangling pointer points to memory which does not belong to current process. e.g. free(ptr) here ptr is dangling pointer. memory leakage means we allocated memory and forgot to deallocate it. this may decrease efficiency

17. What is function pointer? What is its use? Can you write code to declare n define function pointer with & without typedef ?

functionpointer returns address of first instruction of function to which it is pointing. function pointer is used in LSP and IVT. Syntax of functionpointer is void (*ptr)(int, int); and read it as pointer to the block of fun

18. What do you know about heap corruption? Can u explain it using code?

Heap corruption is runtime error which occurs when we try to access memory which does not belong to memory allocated by dynamic memory allocating function like malloc. Stack corruption occurs when we [illegible] memory which is not [illegible]

19. Explain near, far and huge pointers

near ptr is 16 bit pointer to object contained in the current segment (2 byte)

20. What is difference between malloc and calloc?

syntax of malloc() takes one argument and and calloc() takes 2 arguments (size of each element and how many elements) memory allocated by malloc contains garbage and memory allocated by calloc contains 0

21. When we should use array and structure?

We should use array when we want to store data of same datatype e.g. marks of students. And we should use structure when we can want to store logically related items of different datatypes together

22. Why we cannot compare structure objects using == operator?

If our structure size is not in the multiple of word size then compiler adds slack bytes to make it in multiple of word size. == operator does bit by bit checking and checking slack bytes is meaningless

23. What is the difference between structure and union?

Structure size is addition of size of all data members in it and we can access any member at any time. but union size is size of that data member which is largest. we can access only one member of union at a time.

24. What is the difference between string length and string size?

C Language

25. What is variable argument function? Explain all the macros required to implement it with the help of example?

26. Write a swap function 1) Without temporary variable,2) Using macro 3) Using bitwise operator

27. Write a code to allocate and de-allocate memory for multidimensional array dynamically?

28. What do you mean by l-value and R-value?

l means locator value or location value and r means reference value or result value. Rule says that LHS of assignment operator must be l value e.g. b+c = a is not allowed because b has location c has location but b+c has no memory location

29. What is the difference between getch, getche and getchar?

getch() - Reads char from keyboard waits for enter key just get processed after getting any key pressed.
getche() - Same as getch() & echoes char
getchar() - when you press any key these are kept in Buffer after hitting enter first char gets processed & displayed on screen

30. What is file? What is the difference between binary and text mode? List all binary and text mode? List all binary and text modes?

file is collection of data/info in secondary storage device. text mode deals with visual representation of data and binary file deals with memory representation of data. fgetc and fputc, fgets and fputs are text modes, fscanf and fprintf are also text mode. fread() or fwrite() are for binary mode

Su

Data Structure

1. What is stack? Which are the applications of stack?

- Stack is a datastructure which can be implemented using array or linked list. operations
Applications of stack are evaluating infix, prefix or postfix expressions, conversions among them and process stack. When we call main internally it also goes on stack

2. How will you implement two stacks using single array?

We can implement 2 stacks using single array as

3. Convert a given infix expression into postfix and then evaluate it. (Write code).

4. What is Queue? Which are the types of queue?

Queue is datastructure (utility) which can be implemented using array or linked list. Queue can be operated on 2 ends as front and rear. There are 3 types of queues- 1) linear queue, 2) circular queue, 3) priority queue.

5. What is the queue full condition for circular queue?

There are 2 conditions which need to be checked for queue full condition and those are
1) q->front == -1 && q->rear == MAX-1
2) q->front == q->rear && q->front != -1

6. How can we implement stack and queue using linked list?

We can implement stack using linked list with operations ADD_first and Del_first and we can implement stack queue using linked list with operations ADD_last and del_first maintaining 2 pointers (head and tail in singly linked list).

Data Structure

7. Write binary search algorithm with and without recursion.

without recursion

```
while(
{ mid = left + right/2;
  if key == arr[mid]
  return mid;
  if(key < arr[mid])
  right = mid-1;
  else
  left = mid+1;
```

8. Which are the limitations of array over linked list?

We cannot dynamically grow or shrink array like linked list so once we allocate memory it is fixed. If we want to delete any inbetween element then we have to delete it and shift all others, but in linked list we can do this using 2 pointers only.

9. What is linked list?

Linked list is data structure with operations like ADD-last, ADD-FIRST, DEL-FIRST, DEL-LAST etc. We can traverse linked list only through seqial access and we can dynamically grow or shrink memory required for linked list as per requirement

10. What is self-referential structure? What is its need?

structure which contains pointer of its type is called self referential structure. Self referential structure is normally used in programming need where we want to refer next element which is of previous type only i.e. linked list Programming

11. Write a code to reverse linked list?

12. Write a code to reverse linked list using single pointer?

refer pages

Data Structure

13. How can we detect whether linked is linear or circular?

We can detect this in two ways - 1) by traversing whole linked list or by maintaining tail pointer. Complexity of traversal will be O(n) and complexity of maintaining tail pointer will be O(1) so second choice is better

14. How will you write a code to create heterogeneous linked list?

Linked list in which each element or node is of different type is called heterogeneous linked list. ~~example~~ limitation of heterogeneous linked list is that we can only make it for union members.

15. How will you find out address of middle node in single traversal?

We can do this using 2 choices - 1) using count variable or by maintaining 2 pointers. In the first choice we can increment count as we traverse the whole linked list and calculate count/2 to get the mid element or

16. What is hash table? What is hash function? How it is designed?

17. What is collision and how to handle it?

18. What is bucket? What is load factor?

Data Structure

19. Explain tree traversal techniques?

There are 3 kinds of tree traversal techniques- 1) Preorder 2) Inorder 3) postorder we can implement above techniques using recursion or without using recursion. using recursion

20. Which are the types of tree? Explain in brief?

Binary tree, Binary search tree, Complete Binary tree, Strictly binary tree, Full Binary tree, Full Binary tree, Almost complete binary tree, Threaded Binary tree, skewed Binary tree, Balanced binary tree

21. What is AVL tree?

22. What is BFS & DFS?

BFS and DFS are searching techniques used to search node in tree. BFS means Breadth first Search and DFS means Depth first Search. BFS is also called as level wise search

23. Write a code to implement merge sort?

24. Write a code to implement quicksort?

25. What is spanning tree?

26. What is binary search tree?

Binary search tree is a binary tree who in which left child node is always smaller than root node and right child node is always greater than or equal to root node. If we do inorder traversal on binary search

27. Write algorithm to implement TicTacToe game?

28. Write algorithm to implement sudoku game?

29. Write algorithm to implement maze game?

1

2.

3.

4.

5.

6.

Sur

OOP Concepts

1. What do you mean by major and minor pillar/system/element? Which are the major & minor pillars of oops?

There are 4 major and 3 minor pillars of OOPS. 4 major pillars are abstraction encapsulation, modularity and hierarchy. 3 minor pillars are 1) Typing (polymorphism) 2) concurrency, 3) persistence

2. Explain abstraction and encapsulation with the help of real time example?

abstraction means getting essential details only. Encapsulation means hiding details which are unnecessary e.g. lift it's has buttons to operate (i.e. abstraction) and internal details are hidden.

3. What is data security? How can we achieve it in C++?

Giving controlled access to data is called data security. We can declare it as private and give their access to member functions only.

4. What is modularity? What is the advantage of it? How can we achieve it in c++?

Process of dividing ~~code~~ complex system in different files is called modularity. Advantage of modularity is complexity of code decreases. We can achieve this using namespace.

5. What is hierarchy? Which are the types of hierarchies? Explain all with the help of real time example?

Hierarchy means levels or orders of abstraction. main job of hierarchy is to achieve reusability. 1) has a: person has a heart 2) is a: HCl is a chemical 3) use a: 4) creates a: different in village creates current

6. What do you know about polymorphism? With the help of real time example, explain what is need of polymorphism?

polymorphism is ability of object ~~of~~ to take multiple forms. If we do not want to change interface and if we want to add new functionality then use polymorphism. It is used to reduce maintenance ~~s~~ of system e.g. newspapers used to need also and for wrapping items also

OOP Concepts

7. Which are the types of polymorphism and how can we achieve it in C++?

Runtime polymorphism and compile time polymorphism. If call to function gets resolved at runtime then it is Runtime polymorphism and if at compile time then it is compile time polymorphism. we can achieve compile

8. Is it possible to achieve polymorphism in C? if yes then explain it with example?

Yes, we can achieve Runtime Polymorphism in C using function pointer table and (i.e. vtable in C++) and address of specific function gets resolved at runtime

9. What is class and object explain with real time example?

Class is logical entity and object is physical entity. devices class and laptop is object. class is blueprint / Template from which we can create object

10. Which are the characteristics of object?

characteristics of object are state, identity and behaviour. responsibility state of object means value of its data members. member functions of class represent behaviour. unique value used to distinguish between objects.

11. Why size of object of empty class is 1 byte?

class is logical object but object is physical entity. To distinguish it from class compiler vendors allocate 1 byte memory for object of empty class. according to structure size doesn't matter. So compiler vendors did optimization.

12. When we should use inheritance? Which are the types of inheritance?

If we want to add extra functionality to existing class or want to complete partially completed base class then we should use inheritance. Types are 1) single inheritance 2) Multiple inheritance, 3) multilevel 4) Hierarchical

OOP Concepts

13. What is function overloading? Which are the rules to overload function?

If implementation of function is logically equivalent or same then we should give same name to function but signatures should be different. Rules - 1) If type of parameter is same then no. of parameters passed to the function must be different 2) If number of parameters passed to the function are same then type of at least

14. Why return type is not considered in function overloading?

catching values from function is not compulsory. We cannot consider return type in function overloading.

15. When to use abstract class, interface and non-abstract class (With Ref to Inheritance and polymorphism?)

16. What is the difference between interface and implementation inheritance?

interface inheritance means by without changing interface of system we can add extra functionality to system. this is called interface inheritance e.g. product class. Implementation inheritance means if we want to reimplement function of base class into derived class with extra features then it is implementation inheritance

PRASANNA

* time polymorphism using - 1) funct^n overloading
2) operator overloading
3) Template

Runtime Polymorphism is achieved using Function overriding

C++

1. What is the difference between structure and class?

Members of structure are by default public but members of class are by default private.

2. What is "this" pointer? Why static member function does not get this pointer?

this pointer is local pointer, this pointer is link between data member and member function. when we call member function using object then a this pointer is implicitly passed which holds address of current/calling object. this is called this pointer.

3. What is destructor? When we should write destructor inside class?

destructor is used to deinitialize object. we can call destructor explicitly using object only in the case when we use placement new operator.

4. Can we overload destructor? Why?

No. we do not pass any parameter to destructor so we cannot.

5. What is the difference between malloc and new?

malloc() is function	new is operator
malloc() returns NULL if it fails	new throws exception bad_alloc if it fails.
When we use malloc() constructor do not get called	When we use new constructor gets called

6. What is the difference between pointer and reference?

pointer can be initialized to NULL	we cannot initialize reference to NULL
we can create pointer to pointer	we cannot create reference to reference
we can create array of pointers	we cannot create array of references.

7. What is copy constructor? In which condition it gets call?

Copy constructor gets called when we try to assign ~~value of~~ state of ~~already~~ existing object to newly created object.

8. What is shallow copy and deep copy?

Process of assigning value of object to another object is shallow copy. Deep copy needs to be created in 3 conditions 1) If there is pointer data member in class 2) If there is user defined destructor in class and if copy of object need to be created

9. Which are the limitations of operator overloading?

we cannot overload size of, member selection (.), pointer to member selection (.*), conditional operators. Also ~~to~~ meaning of operator can change while operator overloading. we cannot overload static_cast, typeid, dynamic_cast, const_cast & reinterpret_cast

10. In which condition we should overload index operator? Write a code to overload it?

When we want to treat object as array then we should overload index operator

11. What is the difference between copy constructor and assignment operator function?

copy constructor gets called when we assign value of already created object to newly created object and assignment operator function gets called when we try to assign already created object to already created object

12. What is smart pointer? Give any real time exmple of smart pointer?

Smart pointer is object treated as pointer. Iterator is smart pointer which is used to traverse ~~reinterpret cast~~ collection. It is design pattern.

C++

13. What is conversion function? Which are conversion functions in c++?

It is member function of class which is used to convert state of object of fundamental type into user defined type or vice versa following are ~~fun~~ conversion functions - 1) single parameter constructor, 2) Assignment operator fun, 3) Type conversion operator fun

14. Why static member function does not get this pointer?

Static member functions are designed to call with classname so they do not get this pointer

15. Which of the following functions cannot be declared as const? Why?

A. member functions B. global functions C. static functions D. friend functions

We cannot declare global functions and friend functions as const

Member functions in which we try to modify state of object those member functions we cannot declare as const

16. What is singleton class? Write a code for it? Give an example of its usage in program

Singleton class is such class of which we can create only one object Service request class.

17. Write a code to count number of instances created from a class?

We can use counter variable and one static function which will count total no of instances

18. What is diamond problem? How can we overcome it?

In case of diamond inheritance members of indirect base class gets inherited into indirect derived class multiple times that's why we cannot access these members of indirect base class inside member function of indirect derived class directly such problem created by hybrid inheritance is called Diamond Problem

19. What is object slicing? How can we avoid it?

If we try to copy state of object of derived class into state of object of base class then compiler copies values of only base class datamembers which is present inside derived class object into base class object such process is

20. What is virtual function? What is function overriding?

member function of derived class which is designed to call using pointer / Reference of base class is called virtual function of the class. Process of redefining / reimplementing virtual function of base class inside derived class with same signature is called function overriding.

21. Can we declare static member function as a virtual? Why?

Since we cannot declare static member function as virtual, we cannot override static member function in derived class since virtual functions are designed to call using base class pointer or base class reference variable only, and

22. Can we declare constructor as virtual? Why?

No, constructor is not designed to call using pointer or reference. So we cannot declare it as virtual. secondly, if we declare constructor as virtual then its address will be stored in vtable and to store ~~object~~ address of vtable, constructor should be

23. What is need to declare destructor as virtual?

24. What is v-ptr and v-table?

V table will store starting address of all virtual functions of base class. vptr is datamember of derived class & it will store starting address of vptr table. So

size of derived class object gets increased by 4 bytes

C++

25. In C++, by default functions are not virtual, why?

as every function in c++ is not designed to call using pointer or reference. by default, every function is not made virtual in c++. e.g. ~~member fun~~ non-member functions. It can be called using namespacename and scoperesolution operator.

26. What is pure virtual function?

27. How to create final class in C++?

28. When we should declare function as a friend and class as friend?

If most of the ~~member~~ functions inside class are declared as friend of another class then declare it as friend. If less number of functions are declared as friend then do not do whole class as friend.

29. How can we restrict inheritance in C++?

30. What is runtime polymorphism? Explain with real-time example?

When call to function gets resolved at runtime then it is called Runtime Polymorphism.

31. Which are the casting operators given in C++? Explain use of each one with code example.

1) Following are the casting operators in C++: 1) static_cast 2) dynamic_cast 3) reinterpret_cast 4) const_cast. If we want to do type conversion between incompatible types then use ~~static_cast~~ reinterpret_cast

32. Why C++ do not support constructor chaining?

19) object slicing we can avoid object slicing by declaring base class as ~~abstract~~

21) Static member functions are not designed to call using base class pointer or base class reference variable we cannot declare static member function as virtual

22) called, so it will be called two times. and that is not valid

31) If we want to convert pointer to constant object to pointer to nonconstant object then use const cast operator. dynamic cast checks type conversion while doing conversion of incompatible types and overcomes limitation of static_cast So we use it. static_cast operator do not do type conversion and just checks inheritance chain that is its limitation

OS INTERVIEW QUESTIONS

1. Explain OS functionalities in detail.

Process management, memory management File management, CPU scheduling, IO management, Hardware Abstraction, Networking, Protection & security and User interfacing.

2. What is kernel?

The part of OS that performs basic minimal functionalities is called core OS or kernel

3. Explain the output of "ls -l" command?

to see long listing view, we can use ls -l command. for each file each shows read/write permissions, creation date and time, and owner of specified file. Its size and link count are also shown

4. Explain the Process state diagram in detail.

5. Explain Multi-tasking and Multi-Programming.

In Batch system, multiple programs can be loaded in memory. The no. of programs that can be loaded in memory is called degree of multiprogramming. Advantag of multiprogramming is increased CPU utilization. CPU time is shared among multiple processes in the main memory is called multitasking.

6. What is user space and kernel space? What is user mode and kernel mode?

memory area on RAM in which OS is present is called Kernel space and remaining all other area is called User space.

It is invoked
* implementation function from system call table and

OS INTERVIEW QUESTIONS

7. What is system call? How it is called?

System calls are the functions exposed by the kernel so that user programs can access kernel functionalities. System call wrappers / APIs use software interrupt to switch to kernel mode. Then SW int to ISR select appropriate syscall. *

8. Explain arguments and return value of read() and write() system calls.

int ret = read (fd, dest. addr, num of bytes) fd - file descriptor, dest. addr - from which data to be read and num of bytes - how many to read.

int ret = write (fd, src. addr, number of bytes)

9. What is page fault ? What are page replacement algorithms ?

If CPU requests page that is not present in memory then page fault occurs.

FIFO, LRU, Optimal these are page replacement algorithms.

10. What is difference between fork() and exec() ?

fork () is system calls which creates child process by duplicating calling process. exec() is system call which loads new program in the calling process memory

11. What are different IPC mechanisms?

IPC means Interprocess communication. It can be shared memory, message queue, socket, RPC etc.

12. What is pipe?

Pipe is special file which is used to communicate between two processes. It is stream base unidirectional mechanism. Pipe is internally implemented as kernel buffer in which data can be read / written

8) 1st and 3rd argument are same, both return no. of bytes read/write on success and on failure return -1

OS INTERVIEW QUESTIONS

13. What is signal? Explain any five signals and their significance?

OS can send signal to another Process or one Process can send signal to another process. No any data transfer occurs during sending signal.
1) SIGINT - If process do not handle it, it will be terminated. 2) SIGTERM 3) SIGKILL, 4) SIGSTOP. 5) SIGCONT

14. What is interrupt? How it is handled in the system? How it is related to CPU scheduling?

When interrupt occurs then specific address of ISR is taken from IVT and it is executed before going out from ISR, scheduler is called. Scheduler checks if time of current process is completed, If yes then next process is scheduled by using dispatcher.

15. What is context switch?

If interrupt occurs when Process P1 was executing, its execution context was written back into its PCB. After that ISR will be invoked which will call Scheduler and scheduler will decide which process to execute next and will load its execution context in CPU registers.

16. What is vfork() ? and how it differs from fork () ?

vfork() was introduced in BSD unix. It creates PCB for child process and logical copy after that it pauses parent and executes child until exec() comes or exit comes. exec() checks size of a.out and creates actual copy in logical copy

17. What is difference between symbolic link and hard link?

18. What is socket ? How it is created?

Socket is combination of IP address, and Port number. Socket is used for Remote Procedure call. Socket is communication end point. BSD UNIX developed it for first time. 2 types of socket

1) UNIX 2) Internet

TCP UDP

**it has to create its physical copy

PRASANNA WADEKAR

OS INTERVIEW QUESTIONS

19. .What is deadlock? How you will avoid deadlock?

Deadlock occurs when following conditions hold true 1) No preemption, 2) mutual exclusion, 3) Hold & wait 4) circular wait. No preemption means Resource will not be released until task is completed. Mutual exclusion means Resources are *

20. What is Thread Synchronization? Differentiate semaphore and mutex ?

semaphore is synchronization primitive where semaphore may be incremented by one process but another process may decrement it.

In mutex, process which has locked the mutex, same has to unlock it.

21. How will you compose command which will list only "permission attribute" and name of files from directory content?

22. What is difference between process and thread?

1) Process is container which has resource to execute threads.

2) there is parent. child relationship between processes

1) thread, is Executable entity which is present inside process and uses its resources.

2) There is no parent. child relationship between threads.

23.What is Copy on Write?

In Modern Linux, we don't use vfork() but we use fork() itself. Here PCB of child process is created and Page table of parent process is copied in that but actual copy is not created. until child performs on's read operation, all goes well. moment it try to write it * not shareable. Hold & wait means one process has hold resource and it is waiting for other. *

Circular wait means P_1 has hold R_1 and it is waiting for R_2. R_2 is hold by P_2 and it is hold by P_1. we can avoid deadlock using 2 methods

1) advanced announcement of resources required. 2) algorithm - Safe state, Resource allocation graph &

Prasanna Wadekar 9970248407 45

SUNBEAM
Institute of Information Technology
LEARNING INITIATIVE

.NET FRAMEWORK

Session 1:

- **Introduction to .Net Framework**

Objectives of .NET Framework 4.5

The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services.

The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

PRASANNA

.NET Framework Context

Dot Net Framework Components

Development in .NET Framework 4.5

The .NET Framework is a development platform for building apps for Windows, Windows Phone, Windows Server, and Microsoft Azure.

Visual Studio is an Integrated Development Environment.

Application Life Cycle Management (ALM) can be managed using Visual Studio .NET

Design **Develop** **Test** **Debug** **Deploy**

Session 2:

- **Visual Studio .NET 2013 as IDE Tool**

The Design goals are:

Maximize Developer Productivity
Simplify Server based Development
Deliver Powerful Design, Development Tools

RAD (Rapid Application Development Tool) for the next generation internet solutions
Enhanced RAD Support for creating Custom components for Business solutions.

Tools for creating Rich Client Applications
Tools for creating ASP.NET web sites
Tools for Creating S-a-a-S modules
Tools for connecting remote servers
Tools for Cloud Connectivity
Tools for Data Access
Tools for Creating, Developing, Testing and Deploying Solutions.
Help and Documentation
Multiple .NET Language Support

Visual Studio Solutions and Projects

Visual Studio Solution consist of multiple Projects

Visual Studio Project consist of Source code, Resources.

- **C# as .NET Programming Language**

C ++ Heritage
Namespaces, Pointers (in unsafe code),
Unsigned types, etc.
Increased Productivity
Short Learning curve
C# is a type safe Object Oriented Programming Language
C# is case sensitive
Interoperability
C# can talk to COM, DLLs and any of the .NET Framework languages

Structure of first C# program

```
using System;
// A "Hello World!" program in C#
public class HelloWorld
{ public static void Main ()
  {
        Console.WriteLine ("Hello, World");

  }
}
```

Passing Command Line Arguments

```
using System;
/* Invoke exe using console */

public class HelloWorld
{
    public static void Main (string [] args)
    {
        Console.WriteLine ("parameter count = {0}", args.Length);
        Console.WriteLine ("Hello {0}", args [0]);
        Console.ReadLine ();

    }
}
```

Execution of .NET Application

C# code is compiled by CSC.exe (C# compiler) into assembly as Managed code.

Managed code is targeted to Common Language Runtime of .NET Framework

Common Language Runtime converts MSIL code into Platform dependent executable code (native code) for targeted operating System.

Application is executed by Operating System as Process.

C# Types

A C# Program is a collection of types
    Structure, Classes, Enumerations, Interfaces, Delegates, Events
C# provides a set of predefined types
    e.g. int, byte, char, string, object, etc.
Custom types can be created.
All data and code is defined within a type.
    No global variables, no global function.

Types can be instantiated and used by
    Calling methods, setters and getters, etc.
Types can be converted from one type to another.
Types are organized into namespaces, files, and assemblies.
Types are arranged in hierarchy.

In .NET Types are of two categories

Value Type

Directly contain data on Stack.

Primitives:	`int num; float speed;`
Enumerations:	`enum State {off, on}`
Structures:	`struct Point {int x, y ;}`

Reference Types

Contain reference to actual instance on managed Heap.

Root	`Object`
String	`string`
Classes	`class Line: Shape{ }`
Interfaces	`interface IDrawble {….}`
Arrays	`string [] names = new string[10];`
Delegates	`delegate void operation();`

Type Conversion

Implicit Conversion

No information loss

Occur automatically

Explicit Conversion

Require a cast

May not succeed

Information (precision) might be lost

```
int x=543454;
long y=x;        //implicit conversion
short z=(short)x;   //explicit conversion
double d=1.3454545345;
float f= (float) d; //explicit conversion
long l= (long) d     // explicit conversion
```

Constants and read only variables

```
// This example illustrates the use of constant data and readonly fields.

using System;
using System.Text;

namespace ConstData
{

  class MyMathClass
  {
    public static readonly double PI;
    static MyMathClass()
    { PI = 3.14; }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine ("***** Fun with Const *****\n");
      Console.WriteLine ("The value of PI is: {0}", MyMathClass.PI);

      // Error! Can't change a constant!
      // MyMathClass.PI = 3.1444;

      LocalConstStringVariable();
    }

    static void LocalConstStringVariable()
    {
      // A local constant data point.
      const string fixedStr = "Fixed string Data";
      Console.WriteLine(fixedStr);

      // Error!
      //fixedStr = "This will not work!";
    }
  }
}
```

Enumerations

Enumerations are user defined data Type which consist of a set of named integer constants.

```
enum Weekdays { Mon, Tue, Wed, Thu, Fri, Sat}
```

Each member starts from zero by default and is incremented by 1 for each next member.

Using Enumeration Types

```
Weekdays day=Weekdays.Mon;
Console.WriteLine("{0}", day);    //Displays Mon
```

Structures

Structure is a value type that is typically used to encapsulate small groups of related variables.

```
public struct Point
{ public int x;
    public int y;
}
```

Arrays

Declare

```
int [] marks;
```

Allocate

```
int [] marks= new int[9];
```

Initialize

```
int [] marks=new int [] {1, 2, 3, 4, 5, 6, 7, 9};
int [] marks={1,2,3,4,5,6,7,8,9};
```

Access and assign

```
Marks2[i] = marks[i];
```

Enumerate

```
foreach (int i in marks) {Console.WriteLine (i); }
```

Params Keyword

It defines a method that can accept a variable number of arguments.

```
static void ViewNames(params string [] names)
{
 Console.WriteLine ("Names: {0}, {1}, {2}",
                    names [0], names [1], names [2]);
}
public static void Main (string [] args)
{
  ViewNames("Nitin", "Nilesh", "Rahul");
}
```

ref and out parameters

```
void static Swap (ref int n1, ref int n2)
{
     int temp =n1; n1=n2; n2=temp;
}
void static Calculate (float radius, out float area, out float circum)
{
     Area=3.14f * radius * radius;
     Circum=2*3.14f * radius;
}
public static void Main ()
{
     int x=10, y=20;
     Swap(ref x, ref y);
     float area, circum;
     Calculate (5, out area, out circum);
}
```

Session 3:

- Execution Process in .NET Environment

.NET Assembly

A Logical unit of Deployment on .NET Platform.

Components of Assembly

- Manifest
- Metadata
- MSIL code
- Resources

Types of Assemblies

Private Assembly (bin folder)
Shared Assembly (GAC)
System.Data.dll
System.Drawing.dll
System.Web.dll
Etc.

Windows vs. .NET executables

Windows Exe consist of native code
.NET Exe consist of MSIL code

- Inside **.NET Framework**

Common Language Runtime (CLR):

CLR is the heart of the .NET framework and it does 4 primary important things:

1. Garbage collection
2. CAS (Code Access Security)
3. CV (Code Verification)
4. IL to Native translation.

Common Type System (CTS):-

CTS ensure that data types defined in two different languages get compiled to a common data type. This is useful because there may be situations when we want code in one language to be called in other language. We can see practical demonstration of CTS by creating same application in C# and VB.Net and then compare the IL code of both application. Here the data type of both IL code is same.

Common Language Specification (CLS):

CLS is a subset of CTS. CLS is a set of rules or guidelines. When any programming language adheres to these set of rules it can be consumed by any .Net language.

e.g. Lang must be object oriented, each object must be allocated on heap,

Exception handling supported.

Also each data type of the language should be converted into CLR understandable types by the Lang compiler.

All types understandable by CLR forms CTS (common type system) which includes:

```
System.Byte, System.Int16, System.UInt16,
System.Int32, System.UInt32, System.Int64,
System.UInt64, System.Char, System.Boolean, etc.
```

Assembly Loader:

When a .NET application runs, CLR starts to bind with the version of an assembly that the application was built with. It uses the following steps to resolve an assembly reference:

1) Examine the Configuration Files
2) Check for Previously Referenced Assemblies
3) Check the Global Assembly Cache
4) Locate the Assembly through Codebases or Probing

MSIL Code Verification and Just In Time compilation

When .NET code is compiled, the output it produces is an intermediate language (MSIL) code that requires a runtime to execute the IL. So during assembly load at runtime, it first validates the metadata then examines the IL code against this metadata to see that the code is type safe. When MSIL code is executed, it is compiled Just-in-Time and converted into a platform-specific code that's called native code.
Thus any code that can be converted to native code is valid code. Code that can't be converted to native code due to unrecognized instructions is called Invalid code. During JIT compilation the code is verified to see if it is type-safe or not.

Garbage Collection

"Garbage" consists of objects created during a program's execution on the managed heap that are no longer accessible by the program. Their memory can be reclaimed and reused with no adverse effects.

The garbage collector is a mechanism which identifies garbage on the managed heap and makes its memory available for reuse. This eliminates the need for the programmer to manually delete objects which are no longer required for program execution. This reuse of memory helps reduce the amount of total memory that a program needs to run.

In Microsoft's implementation of the .NET framework the garbage collector determines if an object is garbage by examining the reference type variables pointing to it. In the context of the garbage collector, reference type variables are known as "roots". Examples of roots include:

- A reference on the stack
- A reference in a static variable
- A reference in another object on the managed heap that is not eligible for garbage collection
- A reference in the form of a local variable in a method

Generations

Objects that have been around for a long time are fairly unlikely to be garbage, and objects that haven't been around for so long are considerably more likely to be garbage. By and large, this assumption proves to be true for most applications. The garbage collector leverages this fact to improve performance by implementing what are known as generations.

You can think of generations as a way of classifying objects by how long they have been alive. The garbage collector groups objects into three generations, known as generations 0, 1 and 2. Objects in generation 0 have never survived a garbage collection. They're brand new! Objects in generation 1 have survived one garbage collection, and objects in generation 2 have survived two or more collections.

When a garbage collection begins, the garbage collector can pick and choose which generations it wishes to collect. On average, collections in generation 0 will be more effective than those in generation 1, and those in generation 1 will be more effective than those in generation 2. The garbage collector can therefore decide to only collect generation 0, for example, and hopefully reclaim a substantial amount of memory without having to collect the entire managed heap.

IL dis-assembler

The IL Disassembler is a companion tool to the IL Assembler (**Ilasm.exe**). Ildasm.exe takes a portable executable (PE) file that contains intermediate language (IL) code and creates a text file suitable as input to Ilasm.exe.

.NET Framework Folder

C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework

Session 4:

- **Object Orientation in C#**

Object

A real world entity which has well defined structure and behavior.

Characteristics of an Object are:
- State
- Behavior
- Identity
- Responsibility

Pillars of Object Orientation

Abstraction
Encapsulation
Inheritance
Typing, Concurrency, Hierarchy, Persistence

Abstraction

Getting essential characteristic of a System depending on the perspective of on Observer.

Abstraction is a process of identifying the key aspects of System and ignoring the rest

Only domain expertise can do right abstraction.

Abstraction of Person Object
- Useful for social survey
- Useful for healthcare industry
- Useful for Employment Information

Encapsulation

Hiding complexity of a System.
Encapsulation is a process of compartmentalizing the element of an abstraction that constitute its structure and behavior.

Servers to separate interface of an abstraction and its implementation.
User is aware only about its interface: any changes to implementation does not affect the user.

Inheritance

Classification helps in handing complexity.
Factoring out common elements in a set of entities into a general entity and then making it more and more specific.
Hierarchy is ranking or ordering of abstraction.
Code and Data Reusability in System using is a relationship.

Typing

Typing is the enforcement of the entity such that objects of different types may not be interchanges, or at the most, they may be interchanged only in restricted ways.

Concurrency
Different objects responding simultaneously.

Persistence
Persistence of an object through which its existence transcends time and or space.

Namespace and Class

Namespace is a collection .NET Types such as structure, class, interfaces, etc.

```
namespace  EmployeeApp
{
   public class Employee
  {
    private  string empName;
    private  int empID;
    private  float currPay;
    private   int empAge;
    private  string empSSN;
    private static string companyName;

    public Employee ()
    { empID=18; currPay=15000; }
    public Employee (int id, float basicSal)
    { empID=id; currPay= basicSal; }
    public ~Employee()
    { //DeInitializtion }
    public void GiveBonus(float amount)
    {       currPay += amount;     }
```

```
public void DisplayStats()
 {    Console.WriteLine("Name: {0}", empName);
      Console.WriteLine("ID:   {0}", empID);
      Console.WriteLine("Age:  {0}", empAge);
      Console.WriteLine("SSN:  {0}", empSSN);
      Console.WriteLine("Pay:  {0}", currPay);
 }
}
}
```

Partial class

A class can be spread across multiple source files using the keyword partial.

All source files for the class definition are compiled as one file with all class members.

Access modifiers used for defining a class should be consistent across all files.

Properties

Known as smart fields

Special methods that assign and retrieve values from the underlying data members.

Have two assessors:

Get retrieves data member values.

Set enables data members to be assigned

```
public int EmployeeID
{ get {return _id;}
  set {_id=value ;}
}
```

Indexers
Known as smart array.

```
public class Books
{
 private string [] titles= new string [100];
 public string this [int index]
  {
     get{ if (index <0 || index >=100)
                return 0;
          else
                return titles [index];
        }
     set{
          if (! index <0 || index >=100)
                return 0;
          else
                titles [index] =value;
        }
    }
}
public static void Main ()
{ Books mybooks=new Books ();
  Mybooks [3] ="Mogali in Forest";
}
```

Singleton class

```
public class OfficeBoy
 {       private static OfficeBoy _ref = null;
         private  int _val;
         private  OfficeBoy()    {    _val = 10;   }
         public  int Val {    get   {  return _val;   }
                     set { _val = value;   }
                        }
        public static OfficeBoy GetObject()
         {        if (_ref == null)
                  _ref = new OfficeBoy();
                  return _ref;
         }
 }
static void Main(string[] args)
 {     OfficeBoy sweeper,waiter;
       string s1; float f1;
       sweeper = OfficeBoy.GetObject(); waiter = OfficeBoy.GetObject();
       sweeper.Val = 60;
      Console.WriteLine("Sweeper Value : {0}", sweeper.Val);
      Console.WriteLine("Waiter Value  : {0}", waiter.Val);
      s1 = sweeper.Val.ToString();
      f1 = (float)sweeper.Val;
      sweeper.Val = int.Parse(s1);
      sweeper.Val = Convert.ToInt32(s1);
 }
```

- **Arrays**

Multidimensional Arrays (Rectangular Array)

```
int [  , ]  mtrx = new int [2, 3];
     Can initialize declaratively
int [ , ] mtrx = new int [2, 3] { {10, 20, 30}, {40, 50, 60} }
```

Jagged Arrays

An Array of Arrays

```
int [ ]  [ ]  mtrxj = new  int [2] [];
```

Must be initialize procedurally.

Nullable Types

```
class  DatabaseReader
{
  public int? numericValue = null;
  public bool? boolValue = true;
  public int? GetIntFromDatabase()      { return numericValue; }
  public bool? GetBoolFromDatabase()      { return boolValue; }
}
public static void Main (string[] args)
 {
    DatabaseReader  dr = new DatabaseReader();
  // Get int from 'database'.
    int? i = dr.GetIntFromDatabase();
    if (i.HasValue)
      Console.WriteLine("Value of 'i' is: {0}", i.Value);
    else
      Console.WriteLine("Value of 'i' is undefined.");
   // Get bool from 'database'.
    bool? b = dr.GetBoolFromDatabase();
    if (b != null)
      Console.WriteLine("Value of 'b' is: {0}", b.Value);
    else
      Console.WriteLine("Value of 'b' is undefined.");
// If the value from GetIntFromDatabase()is null,assign local variable to 100.
    int? myData = dr.GetIntFromDatabase() ?? 100;
    Console.WriteLine("Value of myData: {0}", myData.Value);
 }
static void LocalNullableVariables ()
{   // Define some local nullable types.
    int? nullableInt = 10;
    double? nullableDouble = 3.14;
    bool? nullableBool = null;
    char? nullableChar = 'a';
    int?[] arrayOfNullableInts = new int?[10];
    // Define some local nullable types using Nullable<T>.
    Nullable<int> nullableInt = 10;
    Nullable<double> nullableDouble = 3.14;
    Nullable<bool> nullableBool = null;
    Nullable<char> nullableChar = 'a';
    Nullable<int>[] arrayOfNullableInts = new int?[10];
   }}}
```

Session 5:

- **Overloading**

Method Overloading

Overloading is the ability to define several methods with the same name, provided each method has a different signature

```
public class MathEngine
{
public static double FindSquare (double number) { // logic defined    }
public static double FindSquare (int number)   { // another logic defined }

}

public static void Main ()
{
     double res= MathEngine.FindSquare(12.5);
     double num= MathEngine.FindSquare(12);

}
```

Operator Overloading

Giving additional meaning to existing operators.
Technique that enhance power of extensibility

```
 public static Complex Operator + (Complex c1, Complex c2)
 {
     Complex temp= new Complex();
     temp.real = c1.real+ c2.real;
     templ.imag = c1.image + c2.imag;
     return temp;
 }
public static void Main ()
{
     Complex o1= new Complex (2, 3);
     Complex o2= new Complex (5, 4);
     Complex o3= 1+ o2;
     Console.WriteLine (o3.real + " " + o3.imag);
}
```

Operator overloading restrictions

Following operators cannot be overloaded.

Conditional logical &&, ||
Array indexing operator []
Cast Operators ()
Assignment operators +=,-=,*=,/+ etc
=,.,? :,->, new, is, sizeof, typeof

The comparison operator, if overloaded, must be overloaded in pairs.

If == is overloaded then != must also be overloaded.

- **C # Reusability**

Inheritance

Provides code reusability and extensibility.

Inheritance is a property of class hierarchy whereby each derived class inherits attributes and methods of its base class.

Every Manager is Employee.

Every Wage Employee is Employee.

```
class Employee
{   public double CalculateSalary ()
       {return basic_sal + hra+ da ;}
}

class Manager: Employee
{
   public double CalculateIncentives ()
    {
     //code to calculate incentives
     Return incentives;
    }
}

static void Main ()
{
     Manager mgr =new Manager ();
     double Inc=mgr. CalculateIncentives ();
     double sal=mgr. CalculateSalary ();
}
```

Constructors in Inheritance

```
class Employee
{ public  Employee ()
    {Console.WriteLine ("in Default constructor") ;}
  public Employee (int eid, ….)
     {Console.WriteLine ("in Parameterized constructor") ;}
}
class Manager: Employee
{
 public Manager (): base ()    {……..}
 public Manager (int id): base (id,….)    {…..}
}
```

Polymorphism

Ability of different objects to responds to the same message in different ways is called Polymorphism.

```
horse.Move();
car.Move();
aeroplane.Move();
```

Virtual and Override

Polymorphism is achieved using virtual methods and inheritance.
Virtual keyword is used to define a method in base class and override keyword is used in derived class.

```
class Employee
{ public virtual double CalculateSalary ()
      {return basic_sal+ hra + da ;}
}
class Manager: Employee
{
  public override double CalculateSalary ()
      {return (basic_sal+ hra + da + allowances) ;}
}

staic void Main ()
{ Employee mgr= new Manager();
  Double salary= mgr. CalculateSalary ();
  Console.WriteLine (salary);
}
```

Shadowing

Hides the base class member in derived class by using keyword new.

```
class Employee
{ public virtual double CalculateSalary ()
     {return basic_sal;}
}
class SalesEmployee:Employee
{ double sales, comm;
  public new double CalculateSalary ()
     {return basic_sal+ (sales * comm) ;}
}
static void Main ()
{ SalesEmployee sper= new SalesEmployee ();
  Double salary= sper.CalculateSalary ();
  Console.WriteLine (salary);
}
```

Sealed class

Sealed class cannot be inherited

```
sealed class SinglyList
{
  public virtual double Add ()
  {// code to add a record in the linked list}
}
public class StringSinglyList:SinglyList
{
  public override double Add ()
  {  // code to add a record in the String linked list}
 }
```

Concrete class vs. abstract classes

Concrete class

Class describes the functionality of the objects that it can be used to instantiate.

Abstract class

Provides all of the characteristics of a concrete class except that it does not permit object instantiation.

An abstract class can contain abstract and non-abstract methods.

Abstract methods do not have implementation.

```
abstract class Employee
 {  public virtual double CalculateSalary();
     { return basic +hra + da ;}
    public abstract double CalculateBonus();
  }
 class Manager: Employee
  {   public override double CalculateSalary();
       { return basic + hra + da + allowances ;}
      public override double CalaculateBonus ()
       { return basic_sal * 0.20 ;}
  }

static void Main ()
{    Manager mgr=new Manager ();
     double bonus=mgr. CalaculateBonus ();
     double Salary=mgr. CalculateSalary ();
}
```

Object class

Base class for all .NET classes

Object class methods

- public bool Equals(object)
- protected void Finalize()
- public int GetHashCode()
- public System.Type GetType()
- protected object MemberwiseClone()
- public string ToString()

Polymorphism using Object

The ability to perform an operation on an object without knowing the precise type of the object.

```
void Display (object o)
{
     Console.WriteLine (o.ToString ());
}
public static void Main ()
{  Display (34);
   Display ("Transflower");
   Display (4.453655);
   Display (new Employee ("Ravi", "Tambade");
}
```

Session 6:

- **Interface Inheritance**

For loosely coupled highly cohesive mechanism in Application.

An interface defines a Contract

Text Editor uses Spellchecker as interfaces.

EnglishSpellChecker and FrenchSpellChecker are implementing contract defined by SpellChecker interface.

```
interface ISpellChecker
{ ArrayList CheckSpelling (string word) ;}
class EnglishSpellChecker:ISpellChecker
{
  ArrayList CheckSpelling (string word)
   {// return possible spelling suggestions}
}
class FrenchSpellChecker:ISpellChecker
{
  ArrayList CheckSpelling (string word)
   {// return possible spelling suggestions}
}
class TextEditor
{
 public static void Main()
 {    ISpellChecker checker= new EnglishSpellChecker ();
      ArrayList words=checker. CheckSpelling ("Flower");
      ...
 }
}
```

Explicit Interface Inheritance

```
interface IOrderDetails     { void ShowDetails() ;}
interface ICustomerDetails { void ShowDetails() ;}
class Transaction: IOrderDetails, ICustomerDetils
{
     void IOrderDetails. ShowDetails()
     {  // implementation for interface IOrderDetails ;}
     void ICustomerDetails. ShowDetails()
     {  // implementation for interface IOrderDetails ;}
}
public static void Main()
{
     Transaction obj = new Transaction();
     IOrderDetails od = obj;
     od.ShowDetails();
     ICustomerDetails cd = obj;
     cd.ShowDetails();
}
```

Abstract class vs. Interface

	Abstract class	Interface
Methods	At least one abstract method	All methods are abstract
Best suited for	Objects closely related in hierarchy.	Contract based provider model
Multiple Inheritance	Not supported	Supported
Component Versioning	By updating the base class all derived classes are automatically updated.	Interfaces are immutable

Building cloned Objects

```
class StackClass: ICloneable
 {  int size; int [] sArr;
    public StackClass (int s) { size=s; sArr= new int [size]; }
    public object Clone()
    {  StackClass s = new StackClass(this.size);
     this.sArr.CopyTo(s.sArr, 0);
     return s;
    }
 }
public static void Main()
{ StackClass stack1 = new StackClass (4);
  Stack1 [0] = 89;
  ……
 StackClass stack2 = (StackClass) stack1.Clone ();
}
```

- **Reflection**

Reflection is the ability to examine the metadata in the assembly manifest at runtime.
Reflection is useful in following situations:

- Need to access attributes in your programs metadata.
- To examine and instantiate types in an assembly.
- To build new types at runtime using classes in System.Reflection.Emit namespace.
- To perform late binding, accessing methods on types created at runtime.

System. Type class
Type class provides access to metadata of any .NET Type.

System. Reflection namespace
Contains classes and interfaces that provide a managed view of loaded types, methods and fields
These types provide ability to dynamically create and invoke types.

Type	Description
Module	Performs reflection on a module
Assembly	Load assembly at runtime and get its type
MemeberInfo	Obtains information about the attributes of a member and provides access to member metadata

Assembly class

```
MethodInfo method;
string methodName;
object result = new object ();
object [] args = new object [] {1, 2};
Assembly asm = Assembly.LoadFile (@"c:/transflowerLib.dll");
Type [] types= asm.GetTypes();
foreach (Type t in types)
{      method = t.GetMethod(methodName);
      ...
      string typeName= t.FullName;
      object obj= asm.CreateInstance(typeName);
       result = t.InvokeMember (method.Name, BindingFlags.Public |
       BindingFlags.InvokeMethod | BindingFlags.Instance, null, obj, args);
      break;
}
string res = result.ToString();
Console.WriteLine ("Result is: {0}", res);
```

MemberInfo class

Base class for all members of the class

`ParameterInfo, FieldInfo, EventInfo, PropertyInfo, MethodInfo, ConstructorInfo`, etc.

Session 7:

.NET Collection Framework

A Collection is a set of similarly typed objects that are grouped together.

System.Array class

The base class for all array Types.

```
int[] intArray= new int[5] { 22,11,33,44,55 };
foreach (int i in intArray )
{ Console.WriteLine( "\t {0}", i); }
Array.Sort(intArray);
Array.Reverse(intArray);
```

Collection Interfaces

Allow collections to support a different behavior

Interface	Description
IEnumertor	Supports a simple iteration over collection
IEnumerable	Supports foreach semantics
ICollection	Defines size, enumerators and synchronization methods for all collections.
IList	Represents a collection of objects that could be accessed by index
IComaprable	Defines a generalized comparison method to create a type-specific comparison
IComparer	Exposes a method that compares two objects.
IDictionary	Represents a collection of Key and value pairs

Implementing IEnumerable Interface

```
public class Team:IEnumerable
{
 private player [] players;
 public Team ()
 {
      Players= new Player [3];
      Players[0] = new Player("Sachin", 40000);
      Players[1] = new Player("Rahul", 35000);
      Players[2] = new Player("Mahindra", 34000);
 }

 public IEnumertor GetEnumerator ()
 {
     Return players.GetEnumerator();
 }
}


public static void Main()
{
Team India = new Team();
foreach(Player c in India)
 {
  Console.WriteLine (c.Name, c.Runs);
 }
}
```

Implementing ICollection Interface

To determine number of elements in a container.
Ability to copy elements into System.Array type

```
public class Team:ICollection
{
 private Players [] players;
 public Team() {.......}
 public int Count {get {return players.Count ;}
}

 // other implementation of Team
}
```

```
public static void Main()
{
Team India = new Team ();
foreach (Player c in India)
 {
  Console.WriteLine (c.Name, c.Runs);
 }
}
```

Implementing IComparable Interface

```
public class Player:IComparable
{

  int IComparable.CompareTo(object obj)
  {
    Player temp= (Player) obj;
    if (this. Runs > temp.Runs)
       return 1;
    if (this. Runs < temp.Runs)
     return -1;
    else
      return 0;
  }
}
public static void Main()
{
 Team India = new Team();

 // add five players with Runs

 Arary.Sort(India);

// display sorted Array

}
```

Using Iterator Method

```
public  class Team
{
 private  player [] players;
 public  Team ()
 {
      Players= new Player [3];
       Players[0] = new Player("Sachin", 40000);
      Players[1] = new Player("Rahul", 35000);
      Players[2] = new Player("Mahindra", 34000);
 }

 public IEnumertor GetEnumerator()
 {
     foreach (Player p in players)
     {yield return p ;}
 }
}

public static void Main()
{  Team India = new Team();
   foreach(Player c in India)
   {
     Console.WriteLine(c.Name, c.Runs);
   }
}
```

- **Collection Classes**

ArrayList class

Represents list which is similar to a single dimensional array that can be resized dynamically.

```
ArrayList countries = new ArrayList ();
countries.Add("India");
countries.Add("USA");
countries.Add("UK");
countries.Add("China");
countries.Add("Nepal");

Console.WriteLine("Count: {0}", countries. Count);

foreach(object obj in countries)
 {
  Console.WriteLine("{0}", obj);

 }
```

Stack class

Represents a simple Last- In- First- Out (LIFO) collection of objects.

```
Stack  numStack = new Stack();
numStack.Push(23);
numStack.Push(34);
numStack.Push(76);
numStack.Push(9);
Console.WriteLine("Element removed", numStack.Pop());
```

Queue class

Represents a first- in, first- out (FIFO) collection of objects.

```
Queue myQueue = new Queue ();
myQueue.Enqueue("Nilesh");
myQueue.Enqueue("Nitin");
myQueue.Enqueue("Rahul");
myQueue.Enqueue("Ravi");

Console.WriteLine("\t Capacity: {0}", myQueue.Capacity);
Console.WriteLine("(Dequeue) \t {0}", myQueue.Dequeue());
```

HashTable class

Represents a collection of Key/ value pairs that are organized based on the hash code of the key.

Each element is a key/ value pair stored in a DictionaryEntry object.

```
Hashtable h = new Hashtable();
h.Add("mo", "Monday");
h.Add("tu", "Tuesday");
h.Add("we", "Wednesday");
IDictionaryEnumerator e= h. GetEnumerator( );
     while(e.MoveNext( ))
     {
       Console.WriteLine(e.Key + "\t" + e.Value);
     }
```

- **Generics**

Are classes, structures, interfaces and methods that have placeholders (type parameters) for one or more of the types they store or use.

```
class HashTable<K,V>
{
 public HashTable();
 public object Get(K);
 public object Add(K, V);
}

HashTable <string,int> addressBook;
...
addressBook.Add("Amit Bhagat", 44235);
..
int extension = addressBook.Get("Shiv Khera");
```

List<T> class

Represents a strongly typed list of objects that can be accessed by index.

Generic equivalent of ArrayList class.

```
List<string> months = new List <string> ();

months.Add("January");
months.Add("February");
months.Add("April");
months.Add("May");
months.Add("June");

foreach(string mon in months)

Console.writeLine(mon);

Months.Insert(2,"March");
```

List of user defined objects

```
class Employee
{
     int eid;
     string ename;
     //appropriate constructor and properties for Employee Entity
}

class EmpComparer:IComparer<Employee>
{
 public int Compare(Employee e1, Employee e2)
  { int ret = e1.Name.Length.CompareTo(e2.Name.Length);
    return ret;
  }
}

public static void Main ()
{
List<Employee>list1 = new List<Employee>();
List1.Add(new Employee(1, "Raghu");
List1.Add(new Employee(2, "Seeta");
List1.Add(new Employee(3, "Neil");
List1.Add(new Employee(4, "Leela");

EmpComparer ec = new EmpComparer();
List1.Sort(ec);
foreach(Employee e in list1)
Console.WriteLine(e.Id + "---------"+ e.Name);
}
```

Prasanna
Wadekar

Stack<T> class

```
Stack<int>numStack = new Stack<int>();

numStack.Push(23);
numStack.Push(34);
numStack.Push(65);

Console.WriteLine("Element removed: ", numStack. Pop());
```

Queue<T> class

```
Queue<string> q = new Queue<string>();
q.Enqueue("Message1");
q.Enqueue("Message2");
q.Enqueue("Message3");

Console.WriteLine("First message: {0}", q.Dequeue());
Console.WriteLine("The element at the head is {0}", q.Peek());
IEnumerator<string> e= q.GetEnumerator();
while(e.MoveNext())
Console.WriteLine(e.Current);
```

LinkedList<T> class

Represents a doubly linked List

Each node is on the type LinkedListNode

```
LinkedList<string> l1= new LinkedList<string>();
L1.AddFirst(new LinkedListNode<string>("Apple"));
L1.AddFirst(new LinkedListNode<string>("Papaya"));
L1.AddFirst(new LinkedListNode<string>("Orange"));
L1.AddFirst(new LinkedListNode<string>("Banana"));

LinkedListNode<string> node=l1.First;
Console.WriteLine(node. Value);
Console.WriteLine(node.Next.Value);
```

Dictionary<K, V> class

Represents a collection of keys and values.

Keys cannot be duplicate.

```
Dictionary<int, string> phones= new Dictionary<int, string>();
phones.Add(1, "James");
phones.Add(35, "Rita");
phones.Add(16, "Meenal");
phones.Add(41, "jim");

phones[16] = "Aishwarya";

Console.WriteLine("Name {0}", phones [12]);
if (!phone.ContainsKey(4))
phones.Add(4,"Tim");
Console.WriteLine("Name is {0}", phones [4]);
```

Session 8:

- **Custom Generic Types**

Generic function

```
public static class MyGenericMethods
{
  public static void Swap<T>(ref T a, ref T b)
  {
   Console.WriteLine ("You sent the Swap() method a {0}", typeof(T));
   T temp; temp = a; a = b; b = temp;
  }

 public static void DisplayBaseClass<T>()
  {Console.WriteLine("Base class of {0} is: {1}.",typeof(T),
                          typeof(T).BaseType);
  }
}

static void Main(string[] args)
  { // Swap 2 ints.
    int a = 10, b = 90;
    Console.WriteLine("Before swap: {0}, {1}", a, b);
    Swap<int>(ref a, ref b);
    Console.WriteLine("After swap: {0}, {1}", a, b);
    Console.WriteLine();

    // Swap 2 strings.
    string s1 = "Hello", s2 = "There";
    Console.WriteLine("Before swap: {0} {1}!", s1, s2);
    Swap<string>(ref s1, ref s2);
    Console.WriteLine("After swap: {0} {1}!", s1, s2);
    Console.WriteLine();

    // Compiler will infer System.Boolean.
    bool b1=true, b2=false;
    Console.WriteLine("Before swap: {0}, {1}", b1, b2);
    Swap (ref b1, ref b2);
    Console.WriteLine("After swap: {0}, {1}", b1, b2);
    Console.WriteLine();
```

```
    // Must supply type parameter if the method does not take params.
    DisplayBaseClass<int>();
    DisplayBaseClass<string>();
  }

static void Swap<T>(ref T a, ref T b)
  {
    Console.WriteLine("You sent the Swap() method a {0}", typeof(T));
    T temp; temp = a; a = b; b = temp;
  }

static void DisplayBaseClass<T>()
 {
   Console.WriteLine("Base class of {0} is: {1}.",
                     typeof(T), typeof(T).BaseType);
 }
```

Custom Structure

```
// A generic Point structure.
public  struct Point<T>
{// Generic state date.
  private T xPos;
  private T yPos;

  // Generic constructor.
  public Point(T xVal, T yVal)
    { xPos = xVal;  yPos = yVal;  }

 // Generic properties.
  public T X
    {  get { return xPos; }  set { xPos = value; }    }
  public T Y
   { get { return yPos; }  set { yPos = value; }    }
  public override string ToString()
  {   return string.Format("[{0}, {1}]", xPos, yPos);   }
  // Reset fields to the default value of the type parameter.
  public void ResetPoint()
  {  xPos = default(T); yPos = default(T);   }
}
```

```
static void Main(string[] args)
 {
   // Point using ints.
   Point<int> p = new Point<int>(10, 10);
   Console.WriteLine("p.ToString()={0}", p.ToString());
   p.ResetPoint();
   Console.WriteLine("p.ToString()={0}", p.ToString());

   // Point using double.
   Point<double> p2 = new Point<double>(5.4, 3.3);
   Console.WriteLine("p2.ToString()={0}", p2.ToString());
   p2.ResetPoint();
   Console.WriteLine("p2.ToString()={0}", p2.ToString());
 }
```

Custom Generic collection class

```
public class Car
  { public string PetName;  public int Speed;
    public Car(string name, int  currentSpeed)
    {     PetName = name;    Speed = currentSpeed;    }
    public Car() { }
  }

public class SportsCar : Car
  {  public SportsCar(string p, int s): base(p, s) { }
     // Assume additional SportsCar methods.
  }
public class MiniVan : Car

  {  public MiniVan(string p, int s)   : base(p, s) { }
     // Assume additional MiniVan methods.
  }

// Custom Generic Collection
  public class CarCollection<T> : IEnumerable<T> where T : Car
  { private List<T>  arCars = new List<T>();
    public T GetCar(int pos) { return arCars[pos]; }
    public void AddCar(T c)  { arCars.Add(c); }
    public void ClearCars()  { arCars.Clear(); }
    public int Count { get { return arCars.Count; }
```

```
// IEnumerable<T> extends IEnumerable,
//therefore we need to implement both versions of GetEnumerator().
    IEnumerator<T> IEnumerable<T>.GetEnumerator()
    {return arCars.GetEnumerator(); }
    IEnumerator IEnumerable.GetEnumerator()
    { return arCars.GetEnumerator(); }
 // This function will only work because of our applied constraint.
   public void PrintPetName(int pos)
   {     Console.WriteLine(arCars[pos].PetName);    }

}
 static void Main(string[] args)
    { // Make a collection of Cars.
      CarCollection<Car>  myCars = new CarCollection<Car>();
      myCars.AddCar(new Car("Alto", 20));
      myCars.AddCar(new Car("i20", 90));
      foreach (Car c in myCars)
      {  Console.WriteLine("PetName: {0}, Speed: {1}",
                                        c.PetName, c.Speed);
      }
     // CarCollection<Car> can hold any type deriving from Car.
      CarCollection<Car>   myAutos = new CarCollection<Car>();
      myAutos.AddCar(new MiniVan("Family Truckster", 55));
      myAutos.AddCar(new SportsCar("Crusher", 40));
      foreach (Car c in myAutos)
       {   Console.WriteLine("Type: {0}, PetName: {1}, Speed: {2}",
                              c.GetType().Name, c.PetName, c.Speed);
       }
    }
```

- **Exceptions Handling**

Abnormalities that occur during the execution of a program (runtime error).

.NET framework terminates the program execution for runtime error.

e.g. divide by Zero, Stack overflow, File reading error, loss of network connectivity

Mechanism to detect and handle runtime error.

```
int a, b=0;

Console.WriteLine("My program starts");

try
 {
     a= 10/b;
 }
catch(Exception e)
{
      Console.WriteLine(e.Message);
}

Console.WriteLine("Remaining program");
```

.NET Exception classes

SystemException	FormatException
ArgymentException	IndexOutOfException
ArgumentNullException	InvalidCastExpression
ArraytypeMismatchException	InvalidOperationException
CoreException	NullReferenceException
DivideByZeroException	OutOfMemoryException
StackOverflowException	

- **User Defined Exception classes**

Application specific class can be created using ApplicationException class.

```
class StackFullException:ApplicationException
{
  public string message;
  public StackFullException(string msg)
  {
     Message = msg;
  }
}

public static void Main(string [] args)
{
  StackClass stack1= new StackClass();

  try
  {  stack1.Push(54);
     stack1.Push(24);
     stack1.Push(53);
     stack1.Push(89);
  }

 catch(StackFullException s)
 {
     Console.WriteLine(s.Message);
 }
}
```

Session 9:

- **Attributes**

Declarative tags that convey information to runtime.

Stored with the metadata of the Element

.NET Framework provides predefined Attributes

The Runtime contains code to examine values of attributes and to act on them

Types of Attributes

Standard Attributes

Custom Attributes

Standard Attributes

.NET framework provides many pre-defined attributes.

General Attributes
COM Interoperability Attributes
Transaction Handling Attributes
Visual designer component- building attributes

```
[Serializable]
public class Employee
{
    [NonSerialized]
    public string name;

}
```

Custom Attributes

User defined class which can be applied as an attribute on any .NET compatibility Types like:

- **Class**
- **Constructor**
- **Delegate**
- **Enum**
- **Event**
- **Field**
- **Interface**
- **Method**
- **Parameter**
- **Property**
- **Return Value**
- **Structure**

Attribute Usage

AttributeUsuageAttribute step 1

It defines some of the key characteristics of custom attribute class with regards to its application , inheritance, allowing multiple instance creation, etc.

```
[AttributeUsuage(AttributeTargets.All, Inherited= false,
AllowMultiple=true)]
```

AttributeUsuageAttribute step 2

Designing Attribute class

Attribute classes must be declared as public classes.
All Attribute classes must inherit directly or indirectly from `System.Attribute`.

```
[AttributeUsuage(AttributeTargets.All, Inherited= false,
AllowMultiple=true)]

public class MyAttribute:System.Attribute

{

    ....

}
```

AttributeUsuageAttribute step 3

Defining Members

Attributes are initialized with constructors in the same way as traditional classes.

```
public MyAttribute (bool myValue)
{
      this.myValue = myValue;
}
```

Attribute properties should be declared as public entities with description of the data type that will be returned.

```
public bool MyProperty
{
    get { return this. MyValue ;}
    set {this. MyValue= value ;}
}
```

Applying Custom Attribute

Custom attribute is applied in following way.

Retrieving Custom Attributes

```
 [Developer("Ravi Tambade", "1"]
public class Employee
{

}
public static void Main()
     {
          Employee emp = new Employee();
          Type t = emp.GetType();
          foreach(Attribute a in t.GetCustomAttributes(true))
          {
           Developer r= (Developer) a;
           //Access r.Name and r.Level
         }
     }
```

- **Delegates**

A delegate is a reference to a method.

All delegates inherits from the `System.delegate` type

It is foundation for Event Handling.

Delegate Types

Unicast (Single cast)
Multicast Delegate

Unicast (Single cast) Delegate

Steps in using delegates

i. Define delegate
ii. Create instance of delegate
iii. Invoke delegate

```
delegate string strDelegate(string str);

strDelegate strDel =new strDelegate(strObject.ReverseStr);

string str=strDel("Hello Transflower");

// or use this Syntax

string str =strDel.Invoke("Hello Transflower");
```

Multicast Delegate

A Multicast delegate derives from `System.MulticastDelegate` class.
It provides synchronous way of invoking the methods in the invocation list.
Generally multicast delegate has void as their return type.

```
delegate void strDelegate(string str);

strDelegate delegateObj;

strDelegate Upperobj = new strDelegate(obj.UppercaseStr);

strDelegate Lowerobj = new strDelegate(obj.LowercaseStr);

delegateObj=Upperobj;

delegateObj+=Lowerobj;

delegateObj("Welcome to Transflower");
```

Delegate chaining

Instances of delegate can be combined together to form a chain

Methods used to manage the delegate chain are

- Combine method
- Remove method

```
//calculator is a class with two methods Add and Subtract
Calculator obj1 = new Calculator();
CalDelegate [] delegates = new CalDelegate[];
          { new CalDelegate(obj1.Add),
           new CalDelegate(Calculator. Subtract)};
CalDelegate chain = (CalDelegate)delegate.Combine(delegates);
Chain = (CalDelegate)delegate.Remove(chain, delegates [0]);
```

Asynchronous Delegate

It is used to invoke methods that might take long time to complete.

Asynchronous mechanism more based on events rather than on delegates.

```
delegate void strDelegate(string str);
public class Handler
{
public static string UpperCase(string s) {return s. ToUpper() ;}
}
strDelegate caller = new strDelegate(handler. UpperCase);
IAsyncResult result = caller.BeginInvoke("transflower", null, null);
// . . .
String returnValue = caller.EndInvoke(result);
```

Anonymous Method

It is called as inline Delegate.

It is a block of code that is used as the parameter for the delegate.

```
delegate string strDelegate(string str);
public static void Main()
{
strDelegate upperStr = delegate(string s) {return s.ToUpper() ;};
}
```

Session 10:

- **Events**

An Event is an automatic notification that some action has occurred.

An Event is built upon a Delegate

```
public delegate void AccountOperation();
public class Account
    {   private int balance;
        public event AccountOperation UnderBalance;
        public  event AccountOperation OverBalance;
        public Account() {balance = 5000 ;}
        public Account(int amount) {balance = amount ;}
        public void Deposit(int amount)
        {  balance = balance + amount;
            if (balance > 100000) {  OverBalance();    }
        }
        public void Withdraw(int amount)
        {   balance=balance-amount;
            if(balance < 5000) { UnderBalance () ;}
        }
    }
}
```

```
class Program
  {  static void Main(string [] args)
        {   Account axisBankAccount = new Account(15000);
            //register Event Handlers
            axisBankAccount.UnderBalance+=PayPenalty;
            axisBankAccount.UnderBalance+=BlockBankAccount;
            axisBankAccount.OverBalance+=PayProfessionalTax;
            axisBankAccount.OverBalance+= PayIncomeTax;
            //Perform Banking Operations
            axisBankAccount.Withdraw(15000);
            Console.ReadLine();
        }
     //Event handlers
      static void PayPenalty()
        {Console.WriteLine("Pay Penalty of 500 within 15 days");     }
      static void BlockBankAccount()
        {Console.WriteLine("Your Bank Account has been blocked") ;}
      static void PayProfessionalTax()
        {Console.WriteLine("You are requested to Pay Professional Tax") ;}
      static void PayIncomeTax()
      {Console.WriteLine("You are requested to Pay Income Tax as TDS") ;}
}
```

Windows Forms

Winforms used to build traditional Desktop applications.

The WinForm API supports RAD (Rapid Application Development) and relies extensively on properties to enable you to tailor the WinForm to your needs.

Microsoft has supplied its default libraries under the **System.Windows.Forms** namespace .The assembly being **System.dll** and **System.Windows.Forms.dll**.

Event Driven Applications

Individual user actions are translated into "events".

Events are passed one by one to application for processing.

GUI based Events

Mouse Move, Mouse Click, Mouse double Click,Key press, Button click, Menu Selection Change in Focus, Window Activation

Code-behind

Events are handled by methods that live behind visual interface

Known as "code-behind"

Developer job is to program these events.

GUI (Graphics User Interface) applications are based on the notion of forms and controls.

A form represents a window

A form contains zero or more controls

First WinForm Application

```
using System.Windows.Forms ;

public class Form1 : Form

{ public static void Main()

  {  //Run the Application

    Application.Run(new Form1());

  }

}
```

GDI +

GDI+ resides in **System.Drawing.dll** assembly.

All GDI+ classes are reside in the

System.Drawing, **System.Text**, **System.Printing**,

System.Internal , **System.Imaging**,
System.Drawing2D and **System.Design** namespaces.

The Graphics Class

The Graphics class encapsulates GDI+ drawing surfaces. Before drawing any object (for example circle, or rectangle) we have to create a surface using Graphics class. Generally we use Paint event of a Form to get the reference of the graphics. Another way is to override OnPaint method.

```
private void form1_Paint(object sender, PaintEventArgs e)
{
Graphics g = e.Graphics;
}
```

Graphics class's methods:

`DrawArc`	Draws an arc from the specified ellipse.
`DrawBezier`	Draws a cubic bezier curve.
`DrawBeziers`	Draws a series of cubic Bezier curves.
`DrawClosedCurve`	Draws a closed curve defined by an array of points.
`DrawCurve`	Draws a curve defined by an array of points.
`DrawEllipse`	Draws an ellipse.
`DrawImage`	Draws an image.
`DrawLine`	Draws a line.
`DrawPath`	Draws the lines and curves defined by a GraphicsPath.
`DrawPie`	Draws the outline of a pie section.
`DrawPolygon`	Draws the outline of a polygon.
`DrawRectangle`	Draws the outline of a rectangle.
`DrawString`	Draws a string.
`FillEllipse`	Fills the interior of an ellipse defined by a bounding rectangle.
`FillPath`	Fills the interior of a path.
`FillPie`	Fills the interior of a pie section.
`FillPolygon`	Fills the interior of a polygon defined by an array of points.
`FillRectangle`	Fills the interior of a rectangle with a Brush.
`FillRectangles`	Fills the interiors of a series of rectangles with a Brush.
`FillRegion`	Fills the interior of a Region.

GDI Objects

After creating a **Graphics** object, you can use it draw lines, fill shapes, draw text and so on. The major objects are:

Brush	Used to fill enclosed surfaces with patterns, colors, or bitmaps.
Pen	Used to draw lines and polygons, including rectangles, arcs, and pies
Font	Used to describe the font to be used to render text
Color	Used to describe the color used to render a particular object. In GDI+ color can be alpha blended

```
Pen pn = new Pen( Color.Blue );

 or

Pen pn = new Pen( Color.Blue, 100 );
```

The Font Class

The **Font class** defines a particular format for text such as font type, size, and style attributes. You use font constructor to create a font.

Initializes a new instance of the **Font** class with the specified attributes.

```
Font font = new Font("Times New Roman", 26);
```

The Brush Class

The **Brush** class is an abstract base class and cannot be instantiated. We always use its derived classes to instantiate a brush object, such as SolidBrush, TextureBrush, RectangleGradientBrush, and LinearGradientBrush.

```
LinearGradientBrush lBrush = new LinearGradientBrush(rect,
Color.Red, Color.Yellow,LinearGradientMode.BackwardDiagonal);

//OR

Brush brsh = new SolidBrush(Color.Red), 40, 40, 140, 140);
```

The SolidBrush class defines a brush made up of a single color. Brushes are used to fill graphics shapes such as rectangles, ellipses, pies, polygons, and paths.

The TextureBrush encapsulates a Brush that uses an fills the interior of a shape with an image.

The LinearGradiantBrush encapsulates both two-color gradients and custom multi-color gradients.

Rectangle Structure

```
public Rectangle(Point, Size);
```

or

```
public Rectangle(int, int, int, int);
```

Point

```
Point pt1 = new Point( 30, 30);
Point pt2 = new Point( 110, 100);
```

Drawing a rectangle

```
protected override void OnPaint(PaintEventArgs pe)
{
Graphics g = pe.Graphics ;
Rectangle rect = new Rectangle(50, 30, 100, 100);
LinearGradientBrush lBrush = new LinearGradientBrush(rect,
Color.Red, Color.Yellow,LinearGradientMode.BackwardDiagonal);
g.FillRectangle(lBrush, rect);
}
```

Drawing an Arc

```
Rectangle rect = new Rectangle(50, 50, 200, 100);
protected override void OnPaint(PaintEventArgs pe)
{
Graphics g = pe.Graphics ;
Pen pn = new Pen( Color.Blue );
Rectangle rect = new Rectangle(50, 50, 200, 100);
g.DrawArc( pn, rect, 12, 84 );
}
```

Drawing a Line

```
protected override void OnPaint(PaintEventArgs pe)
{
Graphics g = pe.Graphics ;
Pen pn = new Pen( Color.Blue );
// Rectangle rect = new Rectangle(50, 50, 200, 100);
Point pt1 = new Point( 30, 30);
Point pt2 = new Point( 110, 100);
g.DrawLine( pn, pt1, pt2 );
}
```

Drawing an Ellipse

```
protected override void OnPaint(PaintEventArgs pe)
{
Graphics g = pe.Graphics ;
Pen pn = new Pen( Color.Blue, 100 );
Rectangle rect = new Rectangle(50, 50, 200, 100);
g.DrawEllipse( pn, rect );
}
```

The FillPath

```
protected override void OnPaint(PaintEventArgs pe)
{
Graphics g = pe.Graphics;
g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
GraphicsPath path = new GraphicsPath(new Point[] {
new Point(40, 140), new Point(275, 200),
new Point(105, 225), new Point(190, 300),
new Point(50, 350), new Point(20, 180), },
new byte[] {
(byte)PathPointType.Start,
(byte)PathPointType.Bezier,
(byte)PathPointType.Bezier,
(byte)PathPointType.Bezier,
(byte)PathPointType.Line,
(byte)PathPointType.Line,
});
PathGradientBrush pgb = new PathGradientBrush(path);
pgb.SurroundColors = new Color[] {
Color.Green,Color.Yellow,Color.Red, Color.Blue,
Color.Orange, Color.White, };
g.FillPath(pgb, path);
}
```

Drawing Text and Strings

```
protected override void OnPaint(PaintEventArgs pe)
{
Font fnt = new Font("Verdana", 16);
Graphics g = pe.Graphics;
g.DrawString("GDI+ World", fnt, new SolidBrush(Color.Red), 14,10);
}
```

- **.NET Data Access**

ADO.NET

A rich set of classes, interface, structures and enumerated types that manage data access from different types of data stores

ADO.NET Features

- A robust connected, disconnected Data Access Model
- Integrated XML support
- Data from varied Data Sources
- Familiarity to ADO programming model (unmanaged environment)
- Enhanced Support

Connected vs. Disconnected Architecture

	Connected	**Disconnected**
State of Connection	Constantly kept Opened	Closed once data is fetched in cache at client side.
Scalability	Limited	More
Current Data	Always available	Not up to date

ADO.NET components

.NET Data Providers

Allow users to interact with different types of data sources.

ODBC Providers

OLEDB Providers

SQL Data Providers

Oracle Data Providers

DataSets

Explicitly designed for disconnected architecture

ADO.NET Interfaces

IDbConnection : Represents an open connection to a data source.

IDbCommand : Represents an SQL statement that is executed while connected to a data source.

IDataReader : Provides a means of reading one or more forward-only streams of result sets obtained by executing a command at a data source.

IDataAdapter: Provides loosely coupled Data Access with Data Sources.

Connection Object

Has the responsibility of establishing connection with the data source.

Connection has to be explicitly closed in finally block.

```
SqlConnection conSql= new SqlConnection();
conSql.ConnectionString = "server=DatabaseServer; Initial Catalog=
Transflower; user id= sa; password=sa";
conSql.Open();
```

Command Object

Used to specify the type of interaction to perform with the database like select, insert, update and delete.

Exposes properties like:

- `CommandText`
- `CommandType`
- `Connection`

Exposes several execute methods like

- `ExecuteScalar()`
- `ExecuteReader()`
- `ExecuteNonQuery()`

```
//Inserting Data
string insertString = "insert into dept (deptId,deptName,loc) values
                       (10,'Mktg','Mumbai')";
string updateString = "update dept set deptName='Marketing' where
                                  deptName='Mktg';
string deleteString = "delete from dept where deptName='ABC'";
conSql.Open();
SqlCommand cmd= new SqlCommand(insertString, conSql);
cmd.ExecuteNonQuery();
```

Getting Single Value

```
SqlCommand cmdSql = new SqlCommand();
cmdSql.Connection = conSql;
cmdSql.CommandText = "Select Count(*) from emp";
int count = (int) cmdSql.ExecuteScalar();
MessageBox.Show(count.ToString());
```

The DataReader Object

Used to only read data in forward only sequential manner.

```
string queryStr = "Select deptName, loc from dept";
conSql.Open();
sqlCommand cmdSql = new SqlCommand(queryStr, conSql);
SqlDataReader dataReader= cmdSql.ExecuteReader();
while(dataReader.Read())
{
      MessageBox.Show("Last Name is "+ dataRead[0].ToString());
      MessageBox.Show("First Name is "+ dataRead[1].ToString());
}
dataReader.Close();
```

Adding parameters to Command

```
conSql.Open();
SqlCommand cmd = new SqlCommand("select * from emp where empNo
                    =@eno", conSql);
SqlParameter param = new SqlParameter();
      param.ParameterName = "@eno";
      param.Value = 100;
      Cmd.Parameters.Add(param);

SqlDataReader reader =cmd.ExecuteDataReader();
while(reader.Read())
{
      //Display data
}
```

Calling a Stored Procedure

```
//Stored Procedure in SQL Server
      CREATE PROCEDURE DeleteEmpRecord (@eno)
      AS delete from emp where empno = @eno;
      RETURN

//C# Code
SqlCommand cmd = new SqlCommand ("DeleteEmpRecord", conSql);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add( new SqlParameter("@eno",100);
```

Multiple Queries

Multiple queries can be executed using a single command object.

```
SqlCommand cmd = new SqlCommand("select * from dept; select * from
                                                emp", conSql);
SqlDataReader reader =cmd.ExecuteDataReader();
  //code to access first result set
bool result = dr.NextResult();
  // code to access next result set
```

Disconnected Data Access

Data Adapter

Represents a set of data commands and a database connection that are used to fill the dataset and update a sql server database.

Forms a bridge between a disconnected ADO.NET objects and a data source.

Supports methods

Fill();

Update ();

```
string sqlStr= "SELECT * FROM Orders";
SqlAdapter da = new SqlAdapter (sqlStr, con);
```

DataAdapter Properties

SelectCommand

Insertcommand

DeleteCommand

UpdateCommand

```
da.SelectCommand.CommandText = "SELECT customerID , ContactName FROM
Customers";
Sqlcommand cmd = new SqlCommand ("INSERT into Customers(CustomerID,
                                 CompanyName) VALUES (898, 'Sunbeam');
da.InsertCommand = command;
```

DataSet

Main object of disconnected architecture

Can store tables with their schema at client side.

Role of CommandBuilder Object

Automatically generates Insert, Update, Delete queires by using SelectCommand property of DataAdapter

```
SqlConnection con = new SqlConnection("server=databaseServer;
            Initial Catalog= Transflower; userid=sa; password=sa");
SqlDataAdapter da= new SqlDataAdapter
                            ("Select * from Customers", con);
SqlCommandBuilder cmdBuilder= new SqlCommandBuilder(da);
DataSet ds= new DataSet();
Da.Fill(ds, "Cusomters");
```

Constraints

Constraints restrict the data allowed in a data column or set of data columns.

Constraint classes in the System.Data namespace

```
UniqueConstraint
ForeignKeyConstraint
```

Using existing primary Key constraint

```
da.FillSchema(ds, schematype.Source, "Customers");
Or
da.MissingSchemaAction = AddWithKey;
da.Fill(ds,"Customers");
```

ADO.NET and XML

With ADO.NET it is easy to

Convert data into XML

Generate a matching XSD schema

Perform an XPath search on a result set.

Interact with an ordinary XML document through the ADO.net

DataSet XML Methods

Getxml()

GetXmlSchema()

ReadXml()

ReadXmlSchema()

WriteXml()

WriteXmlSchema()

InferXmlSchema()

Concurrency in Disconnected Architecture

Disadvantage of disconnected architecture

- Conflict can occur when two or more users retrieve and then try to update data in the same row of a table
- The second use's changes could overwrite the changes made by the first user.

Solutions -

Optimistic concurrency

Retrieves and updates just on row at a time

File Handling and Serialization

Input and Output operation in C#

I/O in C# is stream based.

Stream is flow of data from a source to a receiver through a channel.

Two types of Streams

Byte Streams
Character Streams

Three predefined streams are

Console.Out
Console.In
Console.Error

System.IO Namespace

File IO using System.IO:

System.IO namespace defines all the stream classes.

Directory Class

Static class and helps to manage a single directory.

```
string DirectoryName = @"C:\MyDirectory";
if (Directory.Exists(DirectoryName))
    {MessageBox.Show("Exists"); }
else
    {Directory.CreateDirectory (DirectoryName);
     MessageBox.Show("Created");
    }
```

DirectoryInfo Class

Extends the FileSystemInfo class.

Performing operations such as copying, moving, renaming, creating, and deleting.

```
string DirectoryName = @"C:\MyDirectory";
        if (Directory.Exists(DirectoryName))
           { MessageBox.Show("Exists");     }
        else
          { DirectoryInfo d= new DirectoryInfo (DirectoryName);
            d.Create();
            MessageBox.Show("Created");
          }
```

DriveInfo and Path Class

DriveInfo class

Models a drive and gives its information

```
string s = @"C:\";
DriveInfo d = new DriveInfo(s);
MessageBox.Show (d.AvailableFreeSpace.ToString (), d.Name);
```

Path class

Used to manage file and directory paths.

```
string s = @"C:\temp\MyData.text\machine.config";
MessageBox.Show (Path.GetFileName(s));
MessageBox.Show (Path.GetTempPath ());
```

File Class

Static class and helps to manage a single file

```
string FileName = @"C:\MyFile.dat";
if (File.Exists (FileName))
      {
            MessageBox.Show ("Exists")
      }
else
      {
            File.Create(FileName);
            MessageBox.Show ("Created");
      }
```

Other File System Classes

DeflatesStream
GZipStream
SerialPort
FileSystemInfo

Stream class

Stream is an abstract base class for all other stream classes
Common I/O Stream Classes

FileStream class

Used to perform operations liked read, write, open, and close operations on files on a file system.

For better performance, FileStream class buffers input and output.

BinaryReader and BinaryWriter Class

BinaryReader class is used to read data in binary files.

BinaryWriter class is used to write data to binary files.

```
FileStream fs = newFileStream ( @"C:\FileIO.txt",
FileMode.Create,FileAccess.Write);
BinaryWriter bw = new BinaryWriter (fs);
bw. Write ("Transflower");
bw. Close ();
fs.Close ();
```

The Character Stream Wrapper Classes

TextReader

Abstract class to read sequential series of characters

Inherited by

StreamReader
StringReader

TextWriter

Abstract class to write sequential series of characters

Inherited by

StreamWriter
StringWriter

StreamReader

Used to read data from stream.

```
StreamReader sr = StreamReader (@"c:\FileIO.txt ");
MessageBox.Show (sr.ReadToEnd().ToString());
Sr.Close ();
```

StreamWriter Class

Used to write data to a stream.

```
StreamWriter sw = new StreamWriter (@"C:\FileIO.txt");
sw.Write("Transflower");
sw.Close();
```

Serialization

Persistence is mapped by using serialization.
Serialization is a process of converting data into portable format.

Serialization Process

Type of Serialization

Binary Serialization

System.Runtime.Serialization.Formatters.Binary

XML Serialization

System.Xml.Serialization

SOAP Serialization

System.Runtime.Serialization.Formatters.Soap

[NonSerialized] attribute

Indicates that a field of a serializable class should not be serialized

To reduce the size of the serialized object add the [NonSerialized] attribute to the member

For example

```
[NonSerialized] public string Name;
```

Garbage Collection

COM

Programmatically implement reference counting and handle circular references

C++

Programmatically uses the new operator and delete operator

Visual Basic

Automation memory management

Manual vs. Automatic Memory Management

Common problems with manual memory management

Failure to release memory

Invalid references to freed memory

Automatic memory management provided with .NET Runtime

Eases programming task

Eliminates a potential source of bugs.

Garbage Collector

Manages the allocation and release of memory for your application.

Application Roots.

1. Identifies live object references or application roots and builds their graph
2. Objects not in the graph are not accessible by the application and hence considered garbage.
3. Finds the memory that can be reclaimed.
4. Move all the live object to the bottom of the heap, leaving free space at the top.
5. Looks for contiguous block objects & then shifts the non-garbage objects down in memory.
6. Updates pointers to point new locations.

Generational Garbage collection

Resource Management Types

Implicit Resource Management

With Finalize () method.

Will be required when an object encapsulates unmanaged resources like: file, window or network connection.

Explicit Resource Management

By implementing IDisposable Interface and writing Dispose method.

Implicit Recourse Management with Finalization

Writing Destructors in C#:

```
Class Car
      {
      ~Car () //destructor
                { // cleanup statements}
       }
```

Explicit Resource Management

Implement IDisposable Interface.

Defines Dispose () method to release allocated unmanaged resources.

Multithreading

Multitasking

Ability to have more than one program working at the same time.
The objective to utilize the idle time of the CPU.

Multitasking can be done in 2 ways:

Non Pre-emptive
Pre-emptive

Multithreading

Multithreading is the ability of an operating system to execute different part of a program simultaneously.

When to use multithreading ?

Performing operations that take a large amount of time.
Prioritization of the tasks.
Application has to wait for some event to occur.

Thread Life Cycle

Thread is a path of Execution in a running application.

Thread Life Cycle

System. Threading namespace

Provides a number of classes and other types to support multithreading

Thread Class

```
class ThreadDemo {
private string fName, lName;
 //parameterized constructor defined here
public void Run ()
    {
      Console.WriteLine ("First Name is "+ fName);
      Thread.Sleep (500);
      Console.WriteLine ("Last Name is "+ lName);
       }
}
Class TestThreading
{
 Static void Main ()
 {
  ThreadDemo t1 = new ThreadDemo ("Rahul", "Navale");
  ThreadDemo t2 = new ThreadDemo ("Ravi", "Tambade");
  Thread first = new Thread (new ThreadStart (t1.Run));
  Thread second = new Thread (new ThreadStart (t2.Run));
      first.IsBackground = true;
      second.IsBackground = true;
      first.Start();
      second.Start();
 }
}
```

Thread Scheduling

Threads are schedule for execution using priority.
Thread priorities are defined as ThreadPriority enumeration

Highest
AboveNormal
Normal
BelowNormal
Lowest

```
Thread worker=new Thread(new ThreadStart (StartMethod));
worker.Name = "TransflowerThread";
worker.Priority = ThreadPriority.AboveNormal;
worker.Start();
```

Why Thread Synchronization?

Controlled access to resources needs to be given.
Provided by lock on the object to prevent second thread modifying it.

```
enum operation {credit, debit};
class AccountUser
 {          ...
  BankAccount ac;
   public void run ()
      {
        lock (ac)
         {
             if (operation == Operation.debit)
                 ac.Debit (amt);
            else
                ac.Credit (amt);
         }
      }
}
```

Thread Synchronization

Common problems of multithreading

Deadlocks

Avoided by managing timeouts using threading classes like Monitor class.

Race condition

Avoided by using InterLocked class

Different strategies to synchronize access to instance and static methods and instance fields

Synchronized contexts

Synchronized code regions

Manual synchronization

Synchronization Context

[Synchronization] enables simple, automatic synchronization for instances of the class.
Static fields and methods are not protected from concurrent access by multiple threads

```
using System.Runtime.Remoting.Contexts;
[Synchronization]
class BankAccount
        {
            static int sCount = 0; // multiple threads can access
            int iCount = 0; // only 1 thread can access at a time
            public void Debit (float amt)
                {
                  // … only one thread can access at a time
                }
         }
```

Synchronized Code regions

Restricted access to a block of code, commonly called a critical section is required.

Monitor class

Controls access to these objects by granting a lock on an object to a single thread.
Expose methods like Enter (), Exit (), Wait (), Pulse () and PulseAll().

```
public void Credit ()
{   Monitor.Enter (x);
            try {. . .}
            finally {….}
    Monitor.Exit (x);
```

```
}
```

Manual Synchronization

Access to a variable shared by multiple threads is synchronized manually using InterLocked class.

InterLocked class expose methods like Add (), Increment (), Decrement (), CompareExchange(), etc. to perform atomic operations on such variables.

```
class AccountUser
  {
      static int count; . . .
      public AccountUser () {InterLocked.Increment (ref count);}
     ~AccountUser () {InterLocked.Decrement(ref count); }
   }
```

Use Mutex class

To synchronize between threads across processes Use readWriteLock

In scenarios with x single "writer" and multiple "readers"

```
private static Mutex mutex = new Mutex ();

private static void UseResources()
      {
    Mutex.WaitOne();
   // Place code to access resources her
    Thread.Sleep (500);
    Console.WriteLine ("{0}is leaving the protected area\r\n",
                    Thread.CurrentThread.Name);
    Mutex.ReleaseMutex();
      }
```

Timer Class

Used to periodically execute a method

```
public class Test
{ public static void OnTimer()
   {// background task . . .
   }
   public static void Main ()
   {
     TimerCallback dcallback = new TimerCallback (Test.OnTimer);
     long dTime = 15; //wait before the first tick (in min)
     long pTime = 130;//timer during subsequent invocations (in min)
      Timer timer = new Timer (dcallback, null, dTime, pTime);
     // do something with the timer object Timer.Dispose();
   }
}
```

Thread Pool

Provides a pool of thread that can be used **to post work items, process asynchronous I/O, wait on behalf of other threads, and process timers.**

Used to improve efficiency.

Thread Pooling should not be used when:

Need a task to have a particular priority
Have a task that might run for a long time
Need to place threads into a single-threaded apartment
Need a stable identity to be associated with the thread

Think before Multithreading

- Keeping track of and switching between threads consumes memory resources and CPU time.
- Programming with multiple threads can be complex.
- Shared resources utilization problem.

COM Interoperability

Managed Vs Unmanaged Code

Managed Code

- Targeted to CLR
- Memory managed by Garbage collector
- MSIL code
- .NET complaint

Unmanaged Code

- Code not targeted to CLR
- Memory not managed by Garbage collector
- Operating System complaint

CLR provides two mechanisms for interoperation with unmanaged code.

- Platform Invocation Services
- COM Interoperability

Unmanaged Code – unsafe block

Pointers are used in unsafe block.

Unsafe block

Can be whole class, struct, interface, delegate or any of members of these.

```
public unsafe struct Node
 {
    public int Value;
    public long* Left;
    public long* Top;
 }

unsafe static void main ()
 {
        Point point;
        Point* p = &point;
        p->x = 10;
       p->y = 30;
 }
```

Platform Invoke Service

Used to invoke unmanaged functions (Win32 API) from dlls.

How PInvoke works?

1. Locates the dll containing the required Windows API function.
2. Loads the dll in the memory.
3. Locates the address of the function in the memory and pushes the argument on stack.
4. Transfers the control to unmanaged function for execution.
5. Returns the exception from these functions to be handled by managed code.

Using PInvoke

```
using System.Runtime.InteropServices;
public class TestWin32
{
[DllImport("user32.dll", CharSet=CharSet.Auto,
EntryPoint="MessageBoxA"),CharSet.Ansi,
ExactSpelling=false,CallingConvention.StdCall]
public static extern int MsgBox(int hWnd, string text, string
                                                caption, unit type);
}

private void button3_Click (object sender, EventArgs e)
{
  TestWin.MsgBox(0, "Hello World", "Platform Invoke Sample", 0);
}
```

Interoperability

Reusing COM components into the .NET application irrespective of the technology in which it is developed.

Ensures Installation of the .NET framework does not affect existing COM applications.

COM components and .NET applications can communicate with each other.

COM component vs. .NET components

Component	COM	.NET
Coding model	Interface	Object
Identifier	GUID	Strong name
Compatibility	Binary	Type
Type definition	Type Library	Metadata
Versioning	No	Yes

Using COM component in .NET environment

- Ensure that required COM components is registered with windows registry.
- Add Reference of COM component in .NET application
- Use component by importing namespace as usual.

Using .NET component form COM client

Set the attribute in AssemblyInfo.cs file
[assembly:ComVisible(true)].

Add a reference of .tlb (.NET component).
Use .tlb component like activeX dll.

.NET Remoting

- A mechanism for communicating between objects which are not in the same process.
- Allows Inter Process communication (IPC) as well as Remote Procedural Call (RPC).
- Provides a framework that allows objects to interact with each other across application domains.
- Provides a number of services, including object activation and object lifetime support.

What is Remote Object ?

Any object outside the application domain of the caller application should be considered remote, where the object will be reconstructed. Local objects that cannot be serialized cannot be passed to a different application domain, and are therefore non remotable.

Any object can be changed into a remote object by deriving it from `MarshalByRefObject`, or by making it serializable either by adding the `[Serializable]` tag or by implementing the `ISerializable` interface.

Types of Remotable Object

- Single Call
- Singleton Objects
- Client-Activated Objects (CAO)
-

Domains

In .NET, when an application is loaded in memory, a process is created, and within this process, an application domain is created.

The application is actually loaded in the application domain.

If this application communicates with another application, it has to use Remoting because the other application will have its own domain, and across domains, object cannot communicate directly.

Different application domains may exist in same process, or they may exist in different processes.

Contexts

The .NET runtime further divides the application domain into contexts.

All applications have a default context in which objects are constructed, unless otherwise instructed.

A context, like an application domain, forms a .NET Remoting boundary. Access requests must be marshaled across contexts.

Proxies

When a call is made between objects in the same Application Domain, only a normal local call is required; however, a call across Application Domains requires a remote call. \

In order to facilitate a remote call, a proxy is introduced by the .NET framework at the client side. This proxy is an instance of the `TransparentProxy` class, directly available to the client to communicate with the remote object.

The proxy object ensures that all calls made on the proxy are forwarded to the correct remote object instance.

In .NET Remoting, the proxy manages the marshaling process and the other tasks required to make cross-boundary calls. The .NET Remoting infrastructure automatically handles the creation and management of proxies.

Channels

Channel provides a mechanism by which a stream of bytes is sent from one point to the other (client to server etc.).

> NET Remoting channels existing in `System.Runtime.Remoting.Channels`, the `TcpChannel` and the `HttpChannel`.

Serialization Formatters

.NET Remoting uses serialization to copy marshal-by-value objects, and to send reference of objects which are marshal-by-reference, between application domains.

The .NET Framework supports two kinds of serialization:

Binary serialization and XML serialization.

Formatters are used for encoding and decoding the messages before they are transported by the channel.

There are two native formatters in the .NET runtime:

Binary : `System.Runtime.Serialization.Formatters.Binary`

SOAP: `System.Runtime.Serialization.Formatters.Soap`

How to Implement Remoting

Create a Remotable Object

```
public interface RemoteInterface
 {
    int Sum(int iNum1, int iNum2);
    string Hello(String name);
 }

public class RemoteImpl : MarshalByRefObject, RemoteInterface
    {
      public RemoteImpl()
      { Console.WriteLine("RemoteImpl Object created!"); }

    public int Sum(int iNum1, int iNum2)
      { Console.WriteLine("Sum() called!");
        return iNum1 + iNum2;
      }
    public string Hello(string name)
      {
         Console.WriteLine("Hello() called!");
         return "Hello " + name;
      }
    }
```

Create a Server to Expose the Remotable Object

Server Application configuration File (app.config)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown
          mode="SingleCall"
          type="RemoteImplLib.RemoteImpl, RemoteImplLib"
          objectUri="dac"/>
      </service>
      <channels>
        <channel ref="tcp" port="8467"/>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Server Class

```
class Server
 {
   static void Main(string[] args)
   {  RemotingConfiguration.Configure("Server.exe.config", true);
      Console.WriteLine("Remoting Server Ready!!\n");
      Console.ReadLine();
   }
  }
```

Prasanna Wadekar

Create a Client to Use the Remotable Object

Client Application configuration File

```
<?xml version="1.0"?>
<configuration>
  <system.runtime.remoting>
    <application>
      <client>
        <wellknown type="RemoteImplLib.RemoteImpl, RemoteImplLib"
                   url="tcp://localhost:8467/dac"/>
      </client>
      <channels>
        <channel ref="tcp" port="0"/>
      </channels>
    </application>
  </system.runtime.remoting>
```

Client Class

```
using System.Runtime.Remoting;
using InterfaceLib;
namespace Client
{  class Client
    {
        static void Main(string[] args)
        {
         RemotingConfiguration.Configure("Client.exe.config", true);
         RemoteInterface obj;
         obj = (RemoteInterface)Activator.GetObject(
                         typeof(RemoteInterface),
                        "tcp://localhost:8467/dac");
            Console.WriteLine("Remote Client Ready!\n");
            Console.ReadLine();
            int iRes = obj.Sum(12, 4);
            Console.WriteLine("Sum : " + iRes);
            Console.ReadLine();
            string hello = obj.Hello("DAC");
            Console.WriteLine("Result : " + hello);
        }
    }
}
```

Prasanna wadekar 9970248407

63=00

ASP.NET Web Forms

Background Web Architecture

Background Web Technologies

- Client-side technologies
  - HTML, DHTML, JavaScript
- Server-side technologies
  - ASP (Active Server Pages
- ASP.NET is the next generation of ASP

Background what is ASP?

- Server-side programming technology
- Consists of static HTML interspersed with script
- ASP intrinsic objects (Request, Response, Server, Application, and Session) provide services
- Commonly uses ADO to interact with databases
- Application and session variables
- Application and session begin/end events
- ASP manages threads, database connections...

PRASANNA

Hello World.asp

```
<html>
<head><title>HelloWorld.asp</title></head>
<body>
<form method="post">
<input type="submit" id=button1 name=button1 value="Push Me" />
<%
 if (Request.Form("button1") <> "") then
Response.Write "<p>Hello, the time is " & Now()
end if
%>
</form>
</body>
</html>
```

Background ASP Success

- Simple procedural programming model
- Access to COM components
- ActiveX Data Objects (ADO)
- File System Object
- Custom components
- Script-based: no compiling, just edit, save & run
- VBScript, JScript – leverages existing skills
- Support for multiple scripting languages
- ASP has been very popular

Background ASP Challenges

- Coding overhead (too much code)
  - Everything requires writing code!
- Code readability (too complex; code and UI intermingled)
- Maintaining page state requires more code
- Reuse is difficult
- Supporting many types of browsers is difficult
- Deployment issues (e.g. DLL locking)
- Session state scalability and availability
- Limited support for caching, tracing, debugging, etc.
- Performance and safety limitations of script.

ASP.NET Overview

- ASP.NET provides services to allow the creation, deployment, and execution of Web Applications and Web Services
- Like ASP, ASP.NET is a server-side technology
- Web Applications are built using Web Forms
- Web Forms are designed to make building web-based applications as easy as building Visual Basic applications

ASP.NET Overview Goals

Keep the good parts of ASP and improve the rest
Simplify: less code, easier to create and maintain
Multiple, compiled languages
- Fast
- Scalable
- Manageable
- Available
- Customizable and extensible
- Secure
- Tool support

ASP.NET Overview Key Features

Web Forms
Web Services
Built on .NET Framework
Simple programming model
Maintains page state
Multi-browser support
XCOPY deployment
XML configuration
Complete object model
Session management
Caching
Debugging
Extensibility
Separation of code and UI
Security
ASPX, ASP side by side
Simplified form validation
Cookie less sessions

ASP.NET Overview Demo: Hello World.aspx

```
<%@ Page language="c#" %>
  <html>
     <head></head>
        <script runat="server">
           public void B_Click (object sender, System EventArgs e) {
           Label1.Text = "Hello Transflower, the time is " + DateTime.Now;
           }
        </script>
        <body>
          <form method="post" runat="server">
           <asp:Button onclick="B_Click" Text="Push Me"
                       runat="server" /> <p>
            <asp:Label id=Label1 runat="server" />
          </form>
        </body>
    </html>
```

ASP.NET Overview Architecture

ASP.NET is built upon

- .NET Framework
- Internet Information Server (IIS)
- IIS MMC Snap-In (Internet Services Manager)
  - Tool to manage IIS
- Virtual Directories
  - Provides a mapping between URL and file path
    - E.g., on my machine the URL:
      - http://localhost/TFL15
      - maps to the file path:
      - C :\inetpub\wwwroot\TFL15

Programming Model Controls and Events

- Server-side programming model
- Based on controls and events
- Just like Visual Basic
- Not "data in, HTML out"
- Higher level of abstraction than ASP
- Requires less code
- More modular, readable, and maintainable

Programming Model ASP.NET object Model

User code executes on the web server in page or control event handlers
Controls are objects, available in server-side code
Derived from System.Web.UI.Control
The web page is an object too
Derived from System.Web.UI.Page which is a descendant of System.Web.UI.Control
A page can have methods, properties, etc.

Programming Model Postbacks

A postback occurs when a page generates an
HTML form whose values are posted back to the same page
A common technique for handling form data
In ASP and other server-side technologies the state of the page is lost upon postback...
Unless you explicitly write code to maintain state
This is tedious, bulky and error-prone.

Programming Model Postbacks maintain states

By default, ASP.NET maintains the state of all server-side controls during a postback
Can use method="post" or method="get"
Server-side control objects are automatically populated during postback
No state stored on server
Works with all browsers

Programming Model Server-Side Controls

Multiple sources of controls
Built-in
3rd party
User-defined
Controls range in complexity and power: button, text, drop down, calendar, data grid, ad rotator, validation
Can be populated via data binding.

Programming Model Automatic Browser Compatibility

Controls can provide automatic browser compatibility
Can target Up Level or Down Level browsers

Programming Model Code behind pages

Two styles of creating ASP.NET pages

Controls and code in .aspx file

Controls in .aspx file, code in code-behind page

Supported in Visual Studio.NET

Code-behind pages allow you to separate the user interface design from the code

Allows programmers and designers to work independently.

```
<%@ Codebehind="WebForm1.cs" Inherits=WebApplication1.WebForm1" %>
```

Programming Model Automatic Compilation

Just edit the code and hit the page

ASP.NET will automatically compile the code into an assembly

Compiled code is cached in the CLR Assembly Cache

Subsequent page hits use compiled assembly

If the text of the page changes then the code is recompiled

Works just like ASP: edit, save and run

Programming Basics Page Syntax

The most basic page is just static text

Any HTML page can be renamed .aspx

Pages may contain:

Directives: <%@ Page Language="C#" %>

Server controls: <asp:Button runat="server">

Code blocks: <script runat="server">...</script>

Data bind expressions: <%# %>

Server side comments: <%-- --%>

Render code: <%= %> and <% %>

Use is discouraged; use <script runat=server> with code in event handlers

Instead

Programming Basics Page Directive

Lets you specify page-specific attributes, e.g.

AspCompat: Compatibility with ASP
Buffer: Controls page output buffering
CodePage: Code page for this .aspx page
ContentType: MIME type of the response
ErrorPage: URL if unhandled error occurs
Inherits: Base class of Page object
Language: Programming language
Trace: Enables tracing for this page
Transaction: COM+ transaction setting

Only one page directive per .aspx file

Programming Basics Server Control Syntax

Controls are declared as HTML tags with runat="server" attribute

```
<input type=text id=text2 runat="server" />
<asp:calendar id=myCal runat="server" />
```

Tag identifies which type of control to create
Control is implemented as an ASP.NET class
The id attribute provides programmatic identifier
It names the instance available during postback
Just like Dynamic HTML

Programming Basics Server Control Properties

Tag attributes map to control properties

```
<asp:button id="c1" Text="Foo" runat="server">
<asp:ListBox id="c2" Rows="5" runat="server">
```

Tags and attributes are case-insensitive
Control properties can be set programmatically

```
c1.Text = "Foo";
c2.Rows = 5;
```

Programming Basics Maintaining State

By default. controls maintain their state across multiple postback requests
Implemented using a hidden HTML field: __VIEWSTATE
Works for controls with input data (e.g. TextBox, CheckBox), non-input controls (e.g. Label, DataGrid), and hybrids (e.g. DropDownList, ListBox)

Can be disabled per control or entire page
Set EnableViewState="false"
Lets you minimize size of __VIEWSTATE

Programming Basics Server Code Blocks

Server code lives in a script block marked runat="server"

```
<script language="C#" runat=server>
<script language="VB" runat=server>
<script language="JScript" runat=server>
```

Script blocks can contain
Variables, methods, event handlers, properties
They become members of a custom Page object

Programming Basics Page Events

Pages are structured using events
Enables clean code organization
Avoids the "Monster IF" statement
Less complex than ASP pages
Code can respond to page events
e.g. Page_Load, Page_Unload
Code can respond to control events
Button1_Click
Textbox1_Changed

Programming Basics Page Event Lifecycle

Programming Basics Page Loading

Page_Load fires at beginning of request after controls are initialized
Input control values already populated

```
protected void Page_Load(Object s, EventArgs e)
{
  message.Text = textbox1.Text;
}
```

Page Load fires on every request
Use Page.IsPostBack to execute conditional logic
If a Page/Control is maintaining state then need only initialize it when IsPostBack is false.

```
protected void Page_Load(Object s, EventArgs e)
{
 if (! Page.IsPostBack)
 {
  // Executes only on initial page load
  Message.Text = "Welcome to Transflower!";
 }
 // Rest of procedure executes on every request
}
```

Programming Basics Server Control Event

Change Events
- By default, these execute only on next action event
- E.g. OnTextChanged, OnCheckedChanged
- Change events fire in random order

Action Events
- Cause an immediate postback to server
- E.g. OnClick

Works with any browser
- No client script required, no applets, no ActiveX® Controls!

Programming Basics Wiring up Control Events

Control event handlers are identified on the tag

```
<asp: button onclick="btn1_click"runat=server>
<asp: textbox onchanged="text1_changed"runat=server>
```

Event handler code

```
protected void btn1_Click(Object s, EventArgs e)
 {
  Message.Text = "Button1 clicked";
 }
```

Programming Basics Event Arguments

Events pass two arguments:

The sender, declared as type object
- Usually the object representing the control that generated the event
- Allows you to use the same event handler for multiple controls

Arguments, declared as type EventArgs
- Provides additional data specific to the event
- EventArgs itself contains no data; a class derived from EventArgs will be passed

Programming Basics Page Unloading

Page_Unload fires after the page is rendered
- Don't try to add to output

Useful for logging and clean up

```
protected void Page_Unload(Object s, EventArgs e)
{
  MyApp.LogPageComplete();
}
```

Programming Basics Import Directive
Adds code namespace reference to page
Avoids having to fully qualify .NET types and class names
Equivalent to the C# using directive.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
```

Programming Basics Page Class

The Page object is always available when handling server-side events
Provides a large set of useful properties and methods, including:
Application, Cache, Controls, EnableViewState,
EnableViewStateMac, ErrorPage, IsPostBack, IsValid,
Request, Response, Server, Session,
Trace, User, Validators

DataBind (), LoadControl (), MapPath (), Validate ()

Server Controls
ASP.NET ships with ~50 built-in controls
Organized into logical families
HTML controls
Controls / properties map 1:1 with HTML
Web controls
Richer functionality
More consistent object model

Server Controls HTML Controls
Work well with existing HTML designers
Properties map 1:1 with HTML

```
table.bgcolor ="red";
```

Can specify client-side event handlers
Good when quickly converting existing pages
Derived from System.Web.UI.HtmlControls.HtmlControl
Supported controls have custom class, others derive from HtmlGenericControl
Supported Controls

```
<a>
<img>
<form>
<table>
<tr>
<td>
<th>
<select>
<textarea>
<button>
<input type=text>
<input type=file>
<input type=submit>
<input type=button>
<input type=reset>
<input type=hidden>
```

Can use controls two ways:
Handle everything in action events (e.g. button click)
Event code will read the values of other controls (e.g. text, check boxes, radio buttons, select lists)
Handle change events as well as action events

Server Controls HTML Controls Web Controls
Consistent object model

```
Label1.BackColor = Color.Red;
Table.BackColor = Color.Blue;
```

Richer functionality
E.g. AutoPostBack, additional methods
Automatic uplevel/downlevel support

E.g. validation controls
Strongly-typed; no generic control
Enables better compiler type checking
Web controls appear in HTML markup as namespaced tags
Web controls have an asp: prefix

```
<asp:button onclick="button1_click" runat=server>
<asp:textbox onchanged="text1_changed" runat=server>
```

Defined in the System.Web.UI.WebControls namespace
This namespace is automatically mapped to the asp: prefix
Web Controls provide extensive properties to control display and format, e.g.

Font,
BackColor, ForeColor
BorderColor, BorderStyle, BorderWidth
Style, CssClass
Height, Width
Visible, Enabled

Four types of Web Controls

Intrinsic controls
List controls
Rich controls
Validation controls

Server Controls Intrinsic controls
Correspond to HTML controls
Supported controls

<asp:button>
<asp:imagebutton>
<asp:linkbutton>
<asp:hyperlink>
<asp:textbox>
<asp:checkbox>
<asp:radiobutton>
<asp:image>
<asp:label>
<asp:panel>

<asp:table>

TextBox, ListControl, CheckBox and their subclasses don't automatically do a postback when their controls are changed
Specify AutoPostBack=true to make change events cause a postback

Server Controls List Controls

Controls that handle repetition
Supported controls

<asp:dropdownlist>
<asp:listbox>
<asp:radiobuttonlist>
<asp:checkboxlist>
<asp:repeater>
<asp:datalist>
<asp:datagrid>

Repeater, DataList and DataGrid controls

Powerful, customizable list controls
Expose templates for customization
Can contain other controls
Provide event bubbling through their OnItemCommand event
More about these controls and templates later

Server Controls CheckBoxList and RadioButtonList
Provides a collection of check box or radio button controls
Can be populated via data binding

```
<asp:CheckBoxList id=Check1 runat="server">
 <asp:ListItem>Item 1</asp:ListItem>
 <asp:ListItem>Item 2</asp:ListItem>
 <asp:ListItem>Item 3</asp:ListItem>
 <asp:ListItem>Item 4</asp:ListItem>
 <asp:ListItem>Item 5</asp:ListItem>
</asp:CheckBoxList>
```

Server Control Rich Control
Custom controls with rich functionality
Supported Controls

<asp:calendar>
<asp:adrotator>

More will be added
3rd party controls are coming

Server Control Validation Control

Rich, declarative validation
Validation declared separately from input control
Extensible validation framework
Supports validation on client and server
Automatically detects uplevel clients
Avoids roundtrips for uplevel clients
Server-side validation is always done
Prevents users from spoofing Web Forms

<asp:RequiredFieldValidator>
Ensures that a value is entered
<asp:RangeValidator>
Checks if value is within minimum and maximum values
<asp:CompareValidator>
Compares value against constant, another control or data type
<asp:RegularExpressionValidator>
Tests if value matches a predefined pattern
<asp:CustomValidator>
Lets you create custom client- or server-side validation function
<asp:ValidationSummary>
Displays list of validation errors in one place
Validation controls are derived from System.Web.UI.WebControls.BaseValidator, which is derived from the Label control.

Validation controls contain text which is displayed only if validation fails
Text property is displayed at control location
ErrorMessage is displayed in summary

Validation controls are associated with their target control using the ControlToValidate property

```
<asp:TextBox id=TextBox1 runat=server />
<asp:RequiredFieldValidator id="Req1"
ControlToValidate="TextBox1"
Text="Required Field" runat=server />
```

Can create multiple validation controls with the same target control
Page.IsValid indicates if all validation controls on the page succeed

```
void Submit_click(object s, EventArgs e)
{
 if (Page.IsValid)
 {
  Message.Text = "Page is valid!";
 }
}
```

Display property controls layout

Static: fixed layout, display won't change if invalid

Dynamic: dynamic layout
None: no display; can still use ValidationSummary and Page.IsValid
Type property specifies expected data type: Currency, Date, Double, Integer, String

Data Binding How to Populate Server Controls?
Specify the data in the control's tags
Not dynamic: can't get data from a database
Write code that uses the control's object model
This is okay if you need to populate a simple value or list, but quickly gets too complicated for populating sophisticated displays
Data binding
Create an object that holds the data
(Dataset, Array, string, int, etc.)
Associate that object with the control

Data Binding

Provides a single simple yet powerful way to populate Web Form controls with data
Enables clean separation of code from UI
Supports binding to any data source Properties, expressions, method calls,
Collections (Array, Hashtable, etc.)
Dataset, DataTable, DataView, DataReader
XML
One way snapshot model
Requires code to reapply to data model
Allows you to specify an expression
When the DataBind method of the control is called, the expression is evaluated and bound DataBind for a single control (and subcontrols)
Page.DataBind binds all controls on a page

Works for scalars, e.g. Label control
Works for lists, e.g. DropDown control, ListBox control, etc.
Enables the use of templates

Data Binding Scalar Expressions
Data binding expression: <%# expression %>
Expression is evaluated when DataBind() is called

```
<asp:Label id=label1
Text=<%# "The result is " + (1 + 2) +   ", the time is " +
DateTime.Now.ToLongTimeString() %>  runat="server" />

public void Page_Load(object s, EventArgs e) {
 if (! Page.IsPostBack)
 Page.DataBind();
}
```

Data Binding Simple List
Data binding a list creates a user interface element for each item in the list
Each item contains text (displayed to user) and an optional value (not displayed)

The simple list controls:

```
<asp:ListBox>
Single or multiple select
<asp:DropDownList>
<asp:RadioButtonList>
<asp:CheckBoxList>
```

Steps to data bind a list control
Declare the list control
Optionally set DataValueField and DataTextField
Set its DataSource
Call DataBind() method

Data Binding Database

Data binding can be used to populate server controls with data from a database
Each UI element corresponds to a row
Bind to a DataReader (preferred)
Bind to a DataView of a DataSet
Specify value and text with DataValueField and DataTextField, respectively
Each of these corresponds to a column.

Data Binding Data Source Example

```
DataView GetSampleData()
{
DataSet ds;
SqlConnection cxn;
SqlDataAdapter adp;
cxn = new SqlConnection("server=localhost; " +
"uid=sa;pwd=;database=Northwind");
adp = new SqlDataAdapter("select CategoryID, CategoryName from Categories",
cxn);
ds = new DataSet();
adp.Fill(ds, "Categories");
return ds.Tables["Categories"].DefaultView;
}
```

Data Binding List Binding Example

```
void Page_Load(object s, EventArgs e)
{
ListBox1.DataSource = GetSampleData();
ListBox1.DataValueField = "CategoryID";
ListBox1.DataTextField = "CategoryName";
ListBox1.DataBind();
}

<asp : ListBox id="ListBox1" runat="server" />
void Page_Load(object s, EventArgs e)
{
   ListBox1.DataBind();
}
```

```
<asp:ListBox id="ListBox1" runat="server" DataSource=<%#GetSampleData()%>
DataValueField="CategoryID"
DataTextField="CategoryName" />
```

Data Binding DataGrid
Full-featured list output
Default look is a grid
Default is to show all columns, though you can specify a subset of columns to display
Columns can be formatted with templates
Optional paging
Updateable

Data Binding binding to all Columns
Binding all columns in the datasource
Declare an <asp:DataGrid>
Set its DataSource
Call DataBind()

```
void Page_Load(object s, EventArgs e)
{
            myDataGrid.DataSource = GetSampleData();
            myDataGrid.DataBind();
}
<asp:datagrid id=myDataGrid runat="server" />
```

Data Binding to Specific Column
By default, DataGrid will display all columns
To control columns to display:
Set AutoGenerateColumns="false"
Specify Columns property
Add column definition
BoundColumn
TemplateColumn
ButtonColumn, EditCommandColumn, HyperlinkColumn

Binding to specific columns in the datasource
Declare an <asp:DataGrid>
Declare its Columns collection
Set its DataSource
Call its DataBind() method

```
<asp:datagrid id=myDataGrid
            autogeneratecolumns=false runat=server>
<Columns>
<asp:BoundColumn HeaderText="Id" DataField="title_id" />
<asp:BoundColumn HeaderText="Title" DataField="title"/>
</Columns>
</asp:datagrid>
```

Data Binding DataGrid Paging

When there is too much data to display in one screen, a DataGrid can provide automatic paging
Set AllowPaging="true"
Set PageSize=5
Handle OnPageIndexChanged event
Set page index
Fetch data
Re-bind data

Data Binding Template

Templates provide a powerful way to customize the display of a server control
Customize structure – not just style
Can use controls or other HTML within a template
3rd party controls can expose new templates
With data binding, templates specify a set of markup (HTML or server controls) for each bound piece of data
Not just specifying formatting and style for a column
However, templates are not limited to data binding
No fixed set of templates
Controls may define their own and expose any number of them

Standard templates for list-bound controls

HeaderTemplate: rendered once before all data bound rows
ItemTemplate: rendered once for each row in the data source
AlternatingItemTemplate: like ItemTemplate, but when present is used for every other row
SeparatorTemplate: rendered between each row
FooterTemplate: rendered once, after all data bound rows

Data Binding in Template

Templates need to access the bound data
Container is an alias for the template's containing control
DataItem is an alias for the current row of the datasource
DataBinder.Eval is a utility function provided to retrieve and format data within a template

```
<%# DataBinder.Eval(Container.DataItem, "price", "$ {0}") %>
```

Data Binding Repeater Control

Provides simple output of a list of items
No inherent visual form
Templates provide the visual form
No paging
Can provide templates for separators
Not updateable

```
<asp:Repeater id="repList" runat="server">
 <template name="HeaderTemplate">
   <table>
    <tr><td>Title</td><td>Type</td></tr>
    </template>
    <template name="ItemTemplate">
     <tr>
       <td><%# DataBinder.Eval(Container.DataItem,"title_id") %></td>
       <td><%# DataBinder.Eval(Container.DataItem,"type") %></td>
     </tr>
    </template>
   <template name="FooterTemplate">
 </table>
 </template>
 </asp:Repeater>
```

Data Binding DataList Control

Provides list output with editing
Default look is a table
Customized via templates
Directional rendering (horizontal or vertical)
Single and multiple selection
Alternate item
Updateable
No paging

```
void Page_Load(object s, EventArgs e) {
myDataGrid.DataSource = GetSampleData();
myDataGrid.DataBind();
}
<asp:datalist id=myDataList runat=server>
<template name="itemtemplate">
<b>Title id:</b>
<%# DataBinder.Eval(Container.DataItem, "title_id") %>
<br> <b>Title:</b>
<%# DataBinder.Eval(Container.DataItem, "title") %>
</template>
</asp:datalist>
```

Building Data-Driven ASP.NET Application

Simplified Data Binding

Data binding expressions are now simpler and support hierarchical (XML) data binding

```
<!—classical  ASP.NET 1.x data binding expression -->
<%# DataBinder.Eval (Container.DataItem, "Price") %>
<!-- Equivalent ASP.NET data binding expression -->
<%# Eval ("Price") %>
<!-- XML data binding -->
<%# XPath ("Price") %>
```

Data Source Controls

Declarative (no-code) data binding

Name	Description
SQLDataSource	Connect data-binding controls to SQL database
AccessDataSource	Connect data-binding controls to Access database
XMLDataSource	Connect data-binding controls to XML
ObjectDataSource	Connect data-binding controls to data components
SiteMapDataSource	Connects site navigation controls to site map data

SqlData Source

Declarative data binding to SQL databases
- Any database served by a managed provider

Two-way data binding
- SelectCommand defines query semantics
- InsertCommand, UpdateCommand, and DeleteCommand define update semantics

Optional caching of query results
Parameterized operation

Using SQL Data Source

```
<asp:SqlDataSource ID="Titles" RunAt="server"
  ConnectionString="server=localhost;database=pubs;integrated security=true"
  SelectCommand="select title_id, title, price from titles" />
<asp:DataGrid DataSourceID="Titles" RunAt="server" />
```

Name	Description
ConnectionString	Connection string used to connect to data source
SelectCommand	Command used to perform queries
InsertCommand	Command used to perform inserts
UpdateCommand	Command used to perform updates

DeleteCommand	Command used to perform deletes
DataSourceMode	Specifies whether DataSet or DataReader is used(default=DataSet)
ProviderName	Specifies provider (default=SQL Server .NET provider)

SqlData Source and Caching

SqlDataSource supports declarative caching of results through these properties:

Name	Description
EnableCaching	Specifies whether caching is enabled (default=false)
CacheDuration	Length of time in seconds results should be cached
CacheExpirationPolicy	Specifies whether cache duration is sliding or absolute
CacheKeyDependency	Creates dependency on specified cache key
SqlCacheDependency	Creates dependency on specified database entity

Caching Query Results

```
<asp:SqlDataSource ID="Countries" RunAt="server"
      ConnectionString="server=localhost;database=northwind;..."
      SelectCommand="select distinct country from customers order by country"
      EnableCaching="true" CacheDuration="60" />
      <asp:DropDownList ID="MyDropDownList" DataSourceID="Countries"
      DataTextField="country" AutoPostBack="true" RunAt="server" />
```

Parameterized Command

XxxParameters properties permit database commands to be parameterized

Example: Get value for WHERE clause in SelectCommand from query string parameter or item selected in drop-down list

Example: Get value for WHERE clause in DeleteCommand from GridView

XxxParameter types specify source of parameter values

XxxParameters Properties

Name	Description
SelectParameters	Specifies parameter for select command
InsertParameters	Specifies parameter for insert command

UpdateParameters	Specifies parameter for update command
DeleteParameters	Specifies parameter for delete command
FilterParameters	Specifies parameter for filterExpression

XxxParameters

Name	Description
Parameter	Binds a replaceable parameter to a data field
Controlparameter	Binds a replaceable parameter to a control property
Cookieparameter	Binds a replaceable parameter to a cookie value
FormParameter	Binds a replaceable parameter to a form field
ProfileParameter	Binds a replaceable parameter to a profile property
QueryStringParameter	Binds a replaceable parameter to a query string parameter
SessionParameter	Binds a replaceable parameter to a session variable

Using Control Parameter

```
<asp:SqlDataSource ID="Countries" RunAt="server"
ConnectionString="server=localhost;database=northwind;..."
SelectCommand="select distinct country from customers order by country"/>
<asp:SqlDataSource ID="Customers" RunAt="server"
      ConnectionString="server=localhost;database=northwind;..."
      SelectCommand="select * from customers where country=@Country">
      <SelectParameters>
      <asp:ControlParameter Name="Country" ControlID="MyDropDownList"
                            PropertyName="SelectedValue" />
       </SelectParameters>
</asp:SqlDataSource>
<asp:DropDownList ID="MyDropDownList" DataSourceID="Countries"
       DataTextField="country" AutoPostBack="true" RunAt="server" />
<asp:DataGrid DataSourceID="Customers" RunAt="server" />
```

Calling Stored Procedures

```
<asp:SqlDataSource ID="Countries" RunAt="server"
  ConnectionString="server=localhost;database=northwind;..."
  SelectCommand="proc_GetCountries" />
<asp:SqlDataSource ID="Customers" RunAt="server"
  ConnectionString="server=localhost;database=northwind;..."
  SelectCommand="proc_GetCustomers">
  <SelectParameters>
    <asp:ControlParameter Name="Country" ControlID="MyDropDownList"
      PropertyName="SelectedValue" />
  </SelectParameters>
</asp:SqlDataSource>
<asp:DropDownList ID="MyDropDownList" DataSourceID="Countries"
     DataTextField="country" AutoPostBack="true" RunAt="server" />
<asp:DataGrid DataSourceID="Customers" RunAt="server" />
```

```
CREATE PROCEDURE proc_GetCustomers

@Country nvarchar (32) AS
  SELECT * FROM Customers
  WHERE Country = @Country
GO
CREATE PROCEDURE proc_GetCountries AS
  SELECT DISTINCT Country
  FROM Customers
  ORDER BY Country
GO
```

XMLDataSource

- Declarative data binding to XML data
- Supports caching and XSL transformations
- One-way data binding only; no updating

```
<asp:XmlDataSource ID="Rates" DataFile="Rates.xml" RunAt="server" />

<asp:TreeView ID="MyTreeView" DataSourceID="Rates" RunAt="server" />
```

Key XMLDataSource Properties

Name	Description
DataFile	Path to file containing XML data
TransformFile	Path to file containing XSL style sheet
EnableCaching	Specifies whether caching is enabled (default=false)
CacheDuration	Length of time in seconds data should be cached
CacheExpirationPolicy	Specifies whether cache duration is sliding or absolute
CacheKeyDependency	Creates dependency on specified cache key
XPath	Path expression used to filter data

ObjectDataSource

- Declarative binding to data components
  - Leverage middle-tier data access components
  - Keep data access code separate from UI layer
- Two-way data binding
  - SelectMethod, InsertMethod, UpdateMethod, and DeleteMethod
- Optional caching of query results
- Parameterized operation

Key ODS Properties

Name	Description
TypeName	Type name of data component
SelectMethod	Method called on data component to perform queries
InsertMethod	Method called on data component to perform insert
UpdateMethod	Method called on data component to perform update
DeleteMethod	Method called on data component to perform delete
EnableCaching	Specifies whether caching is enabled(default=false)

Key ODS Properties Count

Name	Description
CacheDomain	Length of time in seconds data should be cached
SqlCacheDependency	Creates dependency on specified database entity
SelectParameter	Specifies parameter for SelectMethod
InsertParameter	Specifies parameter for InsertMethod
UpdateParameter	Specifies parameter for UpdateMethod
DeleteParameter	Specifies parameter for DeleteMethod

Initialization and Clean-Up

ObjectDataSource.SelectMethod et al can identify static methods or instance methods
If instance methods are used:
- ODS creates new class instance on each call
- Class must have public default constructor

Use ObjectCreated and ObjectDisposing events to perform specialized initialization or clean-up work on data components

The GridView Control

- Enhanced DataGrid control
  - Renders sets of records as HTML tables
- Built-in sorting, paging, selecting, updating, and deleting support
- Supports rich assortment of field types, including CheckBoxFields
  - Declared in <Columns> element
- Highly customizable UI

GridView Example

```
<asp:SqlDataSource ID="Employees" RunAt="server"
ConnectionString="server=localhost;database=northwind;..."
 SelectCommand="select lastname, firstname, title from employees" />
<asp:GridView DataSourceID="Employees" Width="100%" RunAt="server" />
```

Output

lastname	firstname	title
Davolio	Nancy	Sales Representative
Fuller	Andrew	Vice President, Sales
Leverling	Janet	Sales Representative
Peacock	Margaret	Sales Representative
Buchanan	Steven	Sales Manager
Suyama	Michael	Sales Representative
King	Robert	Sales Representative
Callahan	Laura	Inside Sales Coordinator
Dodsworth	Anne	Sales Representative

GridView Field Type

Name	Description
BoundField	Renders columns of text fields in data source
ButtonField	Renders columns of button(push button,image,or link
CheckBoxField	Renders Booleans as check boxes
CommandField	Renders columns of selecting and editing GridView data
HyperLinkField	Renders columns of hyperlinks
ImageField	Renders columns of images from URL text
TemplateField	Renders columns using HTML templates

Specifying Field Type

```
      <asp:SqlDataSource ID="Employees" RunAt="server"
      ConnectionString="server=localhost;database=northwind;..."
 SelectCommand="select photo, lastname, firstname, title from  employees" />
      <asp:GridView DataSourceID="Employees" Width="100%" RunAt="server"
      AutoGenerateColumns="false" >
      <Columns>
       <asp:TemplateField HeaderText="Name">
      <ItemTemplate>
        <%# Eval ("firstname") + " " + Eval ("lastname") %>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:BoundField HeaderText="Title" DataField="title" />
  </Columns>
</asp:GridView>
```

Output

The DetailsView Control

- Renders individual records
  - Pair with GridView for master-detail views
  - Or use without GridView to display individual records
- Built-in paging, inserting, updating, deleting
- Uses same field types as GridView
  - Declared in <Fields> element
- Highly customizable UI

DetailsView Example

```
<asp:SqlDataSource ID="Employees" RunAt="server"
ConnectionString="server=localhost;database=northwind;..."
SelectCommand="select employeeid, photo, ... from employees" />
<asp:DetailsView DataSourceID="Employees" RunAt="server"
AllowPaging="true" AutoGenerateRows="false"
PagerSettings-Mode="NextPreviousFirstLast">
<Fields>
<asp:BoundField HeaderText="Employee ID" DataField="employeeid" />
<asp:BoundField HeadorText="Date Hired" DataField="hiredate" />
<asp:TemplateField HeaderText="Name">
<ItemTemplate>
  <%# Eval ("firstname") + " " + Eval ("lastname") %>
</ItemTemplate>
</asp:TemplateField>
<asp:BoundField HeaderText="Title" DataField="title" />
 </Fields>
</asp:DetailsView>
```

Inserting Updating and Deleting

Data controls supply editing UIs
- AutoGenerateXxxButton properties
- Insert/EditRowStyle properties

Data source controls supply editing logic
- Insert/Update/DeleteCommand properties
- Insert/Update/DeleteParameters properties
- Inserting/ed, Updating/ed, Deleting/ed events
- Visual Studio supplies the glue

Editing with GridView

```
<asp:SqlDataSource ID="Employees" RunAt="server"
      ConnectionString="server=localhost;database=northwind;..."
SelectCommand="select employeeid, lastname, firstname from employees"
UpdateCommand="update employees set lastname=@lastname, firstname=
@firstname where employeeid=@original_employeeid">
<UpdateParameters>
<asp:Parameter Name="EmployeeID" Type="Int32" />
<asp:Parameter Name="lastname" Type="String" />
<asp:Parameter Name="firstname" Type="String" />
</UpdateParameters>
</asp:SqlDataSource>
<asp:GridView DataSourceID="Employees" Width="100%" RunAt="server"
DataKeyNames="EmployeeID" AutoGenerateEditButton="true" />
```

Conflict Detection

First-in wins

- Update fails if data has changed
- Structure UpdateCommand accordingly
- Set ConflictDetection="CompareAllValues"

Last-in wins

- Update succeeds even if data has changed
- Structure UpdateCommand accordingly
- Set ConflictDetection="OverwriteChanges"

First-In-Wins Update

```
<asp:SqlDataSource ID="Employees" RunAt="server"
      ConnectionString="server=localhost;database=northwind;..."
      SelectCommand="select employeeid, lastname, firstname from employees"
      UpdateCommand="update employees set lastname=@lastname, firstname=
    @firstname where employeeid=@original_employeeid and lastname=
    @original_lastname and firstname=@original_firstname"
      ConflictDetection="CompareAllValues">
      <UpdateParameters>
    <asp:Parameter Name="EmployeeID" Type="Int32" />
    <asp:Parameter Name="lastname" Type="String" />
    <asp:Parameter Name="firstname" Type="String" />
      </UpdateParameters>
      </asp:SqlDataSource>
      <asp:GridView DataSourceID="Employees" Width="100%" RunAt="server"
      DataKeyNames="EmployeeID" AutoGenerateEditButton="true" />
```

Error Detection

Controls fire events after database updates

- GridView.RowUpdated
- DetailsView.ItemUpdated
- SqlDataSource.Updated, etc.

Event handlers receive "status" objects

- Reveal whether database exception occurred
- Allow exceptions to be handled or rethrown
- Reveal how many rows were affected

Handling Update Errors

```
<asp:SqlDataSource ID="Employees" RunAt="server" ...
  UpdateCommand="..." OnUpdated="OnUpdateComplete">
    ...
</asp:SqlDataSource>
  ...
void OnUpdateComplete (Object source, SqlDataSourceStatusEventsArgs e)
{
    if (e.Exception != null) {
        // Exception thrown. Set e.ExceptionHandled to true to prevent
        // the SqlDataSource from throwing an exception, or leave it set
        // to false to allow SqlDataSource to rethrow the exception
    }
    else if (e.AffectedRows == 0) {
        // No exception was thrown, but no records were updated, either.
        // Might want to let the user know that the update failed
    }
}
```

Managing Look, Feel, and Layout with Visual Studio (ASP.NET)

Master Pages

Master Page Basics

Masters define common content and placeholders (<asp:ContentPlaceHolder>)
Content pages reference masters and fill placeholders with content (<asp:Content>)

Defining a Master Page

```
<%@ Master %>
<html>
  <body>
   <!-- Banner shown on all pages that use this master -->
   <table width="100%">
     <tr>
        <td bgcolor="darkblue" align="center">
          <span style="font-size: 36pt; color: white">ACME Inc.</span>
        </td>
      </tr>
    </table>
    <!-- Placeholder for content below banner -->
    <asp:ContentPlaceHolder ID="Main" RunAt="server" />
  </body>
</html>
```

Applying a Master Page

```
<%@ Page MasterPageFile="~/Site.master" %>
<asp:Content ContentPlaceHolderID="Main" RunAt="server">
  This content fills the place holder "Main" defined in the master page
</asp:Content>
```

Applying a Master Page to a Site

```
<configuration>
  <system.web>
    <pages masterPageFile="~/Site.master" />
  </system.web>
</configuration>
```

Applying a Master Page Programmatically

```
void Page_PreInit (Object sender, EventArgs e)
{
   Page.MasterPageFile = "~/Site.master";
}
```

Default Content

ContentPlaceHolder controls can define content of their own ("default content")
Default content is displayed ONLY if not overridden by content page

```
<%@ Master %>
  ...
<asp:ContentPlaceHolder ID="Main" RunAt="server">
  This is default content that will appear in the absence of a
  matching Content control in a content page
<asp:ContentPlaceHolder>
```

The Page. Master Property

Retrieves reference to master page
Instance of class derived from System.Web.UI.MasterPage
Null if page doesn't have a master

Used to programmatically access content defined in the master page
Use FindControl for weak typing
Use public property in master page for strong typing (preferred)

Accessing a Control in the Master Page (Weak Typing)

In the Master Page
<asp:Label ID="Title" RunAt="server" />

In the Content Page
((Label) Master.FindControl ("Title")).Text = "Orders";

Accessing a Control in the Master Page (Strong Typing)

In the Master Page

```
<asp:Label ID="Title" RunAt="server" />
  <script language="C#" runat="server">
      public string TitleText
      {
            get { return Title.Text; }
            set { Title.Text = value; }
      }
 </script>
```

In the Content page

```
Master.TitleText = "Orders";
```

Nesting Master Pages

Master pages can be nested
Master pages that have masters must contain only Content controls, but Content controls can contain ContentPlaceHolders

```
<!-- Orders.Master -->
<%@ Master MasterPageFile="~/Site.Master" %>
<asp:Content ContentPlaceHolderID="..." RunAt="server">
  <asp:ContentPlaceHolder ID="..." RunAt="server">
    ...
  </asp:ContentPlaceHolder>
<asp:Content>
```

Themes and Skins

Mechanism for theming controls, pages, and sites by group-initializing control properties
Skin = Visual attributes for control(s)
- Physically stored in .skin files
- Default skins and named skins

Theme = Collection of one or more skins
- Physically stored in Themes subfolders
- Global themes and local themes

Applying a Theme to a Page

```
<%@ Page Theme="BasicBlue">
```

Applying a Theme to a Site

```
<configuration>
  <system.web>
    <pages theme="BasicBlue" />
  </system.web>
</configuration>
```

Applying a Theme Programmatically

```
void Page_PreInit (Object sender, EventArgs e)
{
    Page.Theme = "BasicBlue";
}
```

Themes

Defining Skins

```
<!-- Default look for DropDownList controls -->
<asp:DropDownList runat="server" BackColor="hotpink" ForeColor="white" />
<!-- Default look for DataGrid controls -->
<asp:DataGrid runat="server" BackColor="#CCCCCC" BorderWidth="2pt"
  BorderStyle="Solid" BorderColor="#CCCCCC" GridLines="Vertical"
  HorizontalAlign="Left">
  <HeaderStyle ForeColor="white" BackColor="hotpink" />
  <ItemStyle ForeColor="black" BackColor="white" />
  <AlternatingItemStyle BackColor="pink" ForeColor="black" />
</asp:DataGrid>
  ...
```

Named Skins

Skins without SkinIDs are default skins
Skins with SkinIDs are named skins
SkinIDs must be unique per control type
Can be defined in same SKIN file as default skins or in separate files
Use controls' SkinID properties to apply named skins

Defining Named Skins

```
<!-- Default look for DropDownList controls -->

<asp:DropDownList runat="server" BackColor="blue" ForeColor="white"
  SkinID="Blue" />

<!-- Default look for DataGrid conotrols -->

<asp:DataGrid runat="server" BackColor="#CCCCCC" BorderWidth="2pt"
  BorderStyle="Solid" BorderColor="#CCCCCC" GridLines="Vertical"
  HorizontalAlign="Left" SkinID="Blue">
  <HeaderStyle ForeColor="white" BackColor="blue" />
  <ItemStyle ForeColor="black" BackColor="white" />
  <AlternatingItemStyle BackColor="lightblue" ForeColor="black" />
</asp:DataGrid>
...
```

Using a Named Skin

```
      <asp:DropDownList ID="Countries" SkinID="Blue" RunAt="server" />
```

The EnableTheming Property

Supported by all pages and controls
    Defaults to true
Set EnableTheming to false to disable theming for individual controls or entire pages

```
<asp:DropDownList ID="Countries" EnableTheming="false" RunAt="server" />
```

Other ASP.NET Controls

Category	Controls
Data controls	GridView and Details View
Data Source Controls	SqlDataSource,ObjectSource,XmlDataSource, etc
Login controls	Login,CreateUserWizard,PasswordRecovery, etc
Navigation Controls	Menu, TreeView ,and SiteMapPath
Web Part Controls	WebPartManager,WebPartZone, etc
UI Controls	File Upload, Bulleted List , MultiView, Wizard , etc

UI Controls

Name	Description
BulletedList	Renders bulleted lists of items
FileUpload	UI for uploading files to Web Servers
HiddenField	Renders hidden fields
ImageMap	Renders HTML image maps
MultiView	Defines multiple views displayed one at a time
View	Defines views in MultiView controls
Substitution	Designates non-cached regions of cached pages
Wizard	Guides users through stepwise procedures

Building Dynamic Navigation Systems

Site Navigation

- Navigation UIs are tedious to implement
  - Especially if they rely on client-side script
- New controls simplify site navigation
  - TreeView and Menu - Navigation UI
  - SiteMapDataSource - XML site maps
  - SiteMapPath - "Bread crumb" controls
- Public API provides foundation for controls
- Provider-based for flexibility

Site Navigation Schema

TreeView Controls

- Render hierarchical data as trees
  - Expandable and collapsible branches
  - Nodes are navigable, selectable, or static and can include check boxes
- Content defined by TreeNode objects
  - TreeNodes can be added declaratively, programmatically, or through data binding
  - TreeNodes can also be demand-loaded
- Highly customizable UI

Declaring a TreeView

```
<asp:TreeView RunAt="server>
<Nodes>
<asp:TreeNode Text="Training" RunAt="server">
<asp:TreeNode Text="Programming .NET" RunAt="server"
  Navigateurl="Classes.aspx?id=1" />
<asp:TreeNode Text="Programming ASP.NET" RunAt="server"
  NavigateUrl="Classes.aspx?id=2" />
<asp:TreeNode Text="Programming Web Services" RunAt="server"
  NavigateUrl="Classes.aspx?id=3" />
</asp:TreeNode>
<asp:TreeNode Text="Consulting" RunAt="server"
  NavigateUrl="Consulting.aspx" />
<asp:TreeNode Text="Debugging" RunAt="server"
  NavigateUrl="Debugging.aspx" />
</Nodes>
</asp:TreeView>
```

TreeView Properties

Name	Description
checkedNodes	Indicates which, if any, check boxes are checked
ExpandDepth	Specifies the TreeView's initial expand depth
Nodes	TreeNodeCollection representing tree node
SelectedNode	TreeNode representing the node that is currently selected
ShowCheckBoxes	Specifies which, if any, node types include check boxes
ShowExpandCollapse	Specifies whether expand/collapse indicators are shown
ShowLines	Specifies whether nodes are connected by lines

Key TreeView Properties Count

Name	Description
LevelStyle	Specifies appearance of nodes by level
NodeStyle	Specifies default appearance of nodes
RootNodeStyle	Specifies appearance of root nodes
LeafNodeStyle	Specifies appearance of leaf nodes
SelectedNodeStyle	Specifies appearance of selected nodes
HoverNodeStyle	Specifies appearance of nodes when cursor hovers overhead
ImageUrl properties	Identify expand/collapse indicator images and other

Key TreeNode Properties

Name	Description
Checked	Indicates whether the nodes check box is selected
ImageUrl	URL of image to display next to node text
NavigateUrl	URL to navigate to when node is clicked
SelectAction	Action to take when non-navigable node is clicked
Selected	Indicates whether node is currently selected
ShowCheckBox	Specifies whether node includes a check box
Test	Node text

Example

```
<asp:TreeView ShowLines="true" Font-Name="Verdana" Font-Size="10pt" ... >
<SelectedNodeStyle BackColor="Yellow" />
<HoverNodeStyle BackColor="LightBlue" />
<Nodes>
<asp:TreeNode Text="Not selectable" SelectAction="None" RunAt="server">
<asp:TreeNode Text="Selectable" SelectAction="Select" RunAt="server" >
<asp:TreeNode Text="Click to expand or collapse"
  SelectAction="Expand" Runat="server">
<asp:TreeNode Text="Click to select and expand or collapse"
  SelectAction="SelectExpand" Runat="server">
<asp:TreeNode Text="Check box node" ShowCheckBox="true"
  Runat="server">
<asp:TreeNode Text="Click to navigate" NavigateUrl="..."
  Runat="server" /></asp:TreeNode>
</asp:TreeNode> </asp:TreeNode> </asp:TreeNode></asp:TreeNode>
</Nodes>
</asp:TreeView>
```

Tree View Events

Name	Description
checkedChanged	Fired when a check box is checked or unchecked
SelectedNodeChanged	Fired when the selected node changes
TreeNodeCollapsed	Fired when a branch is collapsed
TreeNodeExpanded	Fired when a branch is expanded
TreeNoeDataBound	Fired when a tree node binds to a data source
TreeNodePopulate*	Fired when a PopulateOnDemand node needs content

Using SelectedNode Changed

```
<asp:TreeView ID="Tree" OnSelectedNodeChanged="OnUpdate" RunAt="server">
<Nodes>
   ...
</Nodes>
</asp:TreeView>
  .
  .
  .
<script language="C#" runat="server">
void OnUpdate (Object sender, EventArgs e)
{
    // Get the text of the selected node
    string text = Tree.SelectedNode.Text;
      ...
}
</script>
```

Populating Nodes on Demand

```
<asp:TreeView OnTreeNodePopulate="OnPopulate" EnableClientScript="false"
  RunAt="server">
  <Nodes>
    <asp:TreeNode Text="Populate this node on demand"
      PopulateOnDemand="true" RunAt="server" />
  </Nodes>
</asp:TreeView>
  .
  .
  .
<script language="C#" runat="server">
void OnPopulate (Object sender, TreeNodeEventArgs e)
{
    // Called first time the populate-on-demand node is expanded
    TreeNode node = new TreeNode ("This node added dynamically");
    e.Node.ChildNodes.Add (node);
}
</script>
```

Menu Controls

Drop-down/fly-out menus for Web pages
Items are navigable or selectable
Can be oriented horizontally or vertically
Content defined by MenuItem objects
MenuItems can be added declaratively, programmatically, or through data binding
Highly customizable UI

Declaring a Menu

```
<asp:Menu Orientation="Horizontal" RunAt="server">
<Items>
<asp:MenuItem Text="Training" RunAt="server">
<asp:MenuItem Text="Programming .NET" RunAt="server"
  Navigateurl="Classes.aspx?id=1" />
<asp:MenuItem Text="Programming ASP.NET" RunAt="server"
  NavigateUrl="Classes.aspx?id=2" />
<asp:MenuItem Text="Programming Web Services" RunAt="server"
  NavigateUrl="Classes.aspx?id=3" />
</asp:MenuItem>
<asp:MenuItem Text="Consulting" RunAt="server"
  NavigateUrl="Consulting.aspx" />
<asp:MenuItem Text="Debugging" RunAt="server"
  NavigateUrl="Debugging.aspx" />
</Items>
</asp:Menu>
```

Key Menu Properties

Name	Description
Items	MenuItemCollection representing menu items
ItemWrap	Specifies whether menu item should wrap
Orientation	Specifies whether to orient menu horizontally or vertically
SelectedItem	MenuItem representing the item that is currently selected
StaticStyle properties	Specify appearance of static menus
DynamicStyle properties	Specify appearance of dynamic menus

Key MenuItem Properties

Name	Description
ImageUrl	URL of image to display next to menu item text
NavigateUrl	URL to navigate to when menu item is clicked
Selected	Indicates whether the item is currently selected
Text	Menu item text (visible to user)
ToolTip	Tooltip text displayed when cursor pauses over menu item
Value	Menu item value (not visible to user)

Menu Events

Name	Description
MenuItemClick	Fired when a menu item is clicked
MenuItemDataBound	Fired when a menu item binds to a data source

Using MenuItemClick

```
<asp:Menu ... OnMenuItemClick="OnClick" RunAt="server">
<Items>
...
</Items>
</asp:Menu>
 .
 .
 .
<script language="C#" runat="server">
 void OnClick (Object sender, MenuEventArgs e)
 {
  // Get the text of the selected menu item
  string text = e.Item.Text;
  ...
 }
</script>
```

SiteMapDataSource

Data source control representing site maps
Site map = List of pages and URLs
Nodes can include descriptive text
Permits TreeViews and Menus to be populated with links through data binding
Supports "security trimming"
Specified nodes visible only to specified roles
Provider-based for flexible data storage

XML Site Map

```
<siteMap>
  <siteMapNode title="Home" description="" url="default.aspx">
  <siteMapNode title="Training" url="Training.aspx"
   description="Training for .NET developers">
  <siteMapNode title="Programming .NET" url="Classes.aspx?id=1"
   description="All about the .NET Framework" />
  <siteMapNode title="Programming ASP.NET" url="Classes.aspx?id=2"
   description="All about ASP.NET" />
  <siteMapNode title="Programming Web Services" url="Classes.aspx?id=3"
   escription="All about Web services" />
  </siteMapNode>
  <siteMapNode title="Consulting" url="Consulting.aspx"
description="Consulting for .NET projects" />
  <siteMapNode title="Debugging" url="Debugging.aspx"
description="Help when you need it the most" />
  </siteMapNode>
</siteMap>
```

Tree View and Site Maps

```
<asp:SiteMapDataSource ID="SiteMap" RunAt="server" />
<asp:TreeView DataSourceID="SiteMap" RunAt="server" />
```

Menus and Site Maps

```
<asp:SiteMapDataSource ID="SiteMap" RunAt="server" />
<asp:Menu DataSourceID="SiteMap" RunAt="server" />
```

Changing the file Name

```
<configuration>
  <system.web>
   <siteMap>
     <providers>
       <remove name="AspNetXmlSiteMapProvider" />
<add name="AspNetXmlSiteMapProvider"
     type="System.Web.XmlSiteMapProvider, System.Web, ..."
     siteMapFile="Acme.sitemap" />
     </providers>
   </siteMap>
 </system.web>
</configuration>
```

<siteMapNode> Attributes

Name	Description
description	Description of node
role	Role or roles for which this node is visible
title	Title of this node
url	URL of this node

Security trimming

Visible to everyone

```
<siteMap>
  <siteMapNode title="Home" description="" url="default.aspx">
    <siteMapNode title="Announcements" url="Announcements.aspx"
      description="Information for all employees" />
    <siteMapNode title="Salaries" url="Salaries.aspx"
      description="Salary data" roles="Managers,CEOs" />
  <siteMapNode>
</siteMap>
```

Only visible to CEO and managers

Enabling Security trimming

```
<configuration>
  <system.web>
    <siteMap>
      <providers>
        <remove name="AspNetXmlSiteMapProvider" />
        <add name="AspNetXmlSiteMapProvider"
          type="System.Web.XmlSiteMapProvider, System.Web, ..."
          securityTrimmingEnabled="true"
          siteMapFile="web.sitemap" />
      </providers>
    </siteMap>
  </system.web>
</configuration>
```

SiteMapDataSource Properties

Name	Description
Provider	Provider used to obtain site map data
SiteMapProvider	Name of provider used to obtain site map data
ShowStartingNode	Specifies whether to show the root node
StarFormCurrentNode	Specifies whether starting node should be the root node (false) or the current node (true). Default=false
StartingNodeOffset	Starting node identified by level (default=0)
StartingNodeURL	Starting node identified by url

Handling the Root Site Map Node

```
<asp:SiteMapDataSource ID="SiteMap" ShowStartingNode="false"
RunAt="server" />
<asp:TreeView DataSourceID="SiteMap" RunAt="server" />
```

SiteMapPath Control

"Bread crumbs" showing path to page
- By default, renders current node as static text
- By default, renders parent nodes as hyperlinks
- Highly customizable UI
  - Nodes can be stylized and templatized
  - Separators can be stylized and templatized
- Integrates with site map providers to acquire and path info

Using SiteMapPath

```
<asp:SiteMapPath RunAt="server" />
```

Home > Training > Programming ASP.NET

Key SiteMapPath Properties

Name	Description
CurrentNodeStyle	Style used to render the current node
CurrentNodeTemplate	HTML template used to render the current node
NodeStyle	Style used to render non-current nodes
NodeStyleTemplate	HTML template used to render non-current nodes
PathSeparator	Text used for node separators (default=">")
PathSeparatorStyle	Style used to render node separators
PathSeparatorTemplate	HTML template used to render node separators

Stylizing SiteMapPath

```
<asp:SiteMapPath Font-Name="Verdana" Font-Size="10pt" RunAt="server">
  <CurrentNodeStyle Height="24px" BackColor="Yellow" Font-Bold="true" />
  <NodeStyle Height="24px" />
  <PathSeparatorTemplate>
    <ItemTemplate>
      <asp:Image ImageUrl="~/images/arrow.gif" RunAt="server" />
    </ItemTemplate>
  </PathSeparatorTemplate>
</asp:SiteMapPath>
```

SiteMap Providers

- Site maps are provider-based
  - Provider interprets site map data and provides it to SiteMapDataSource controls
  - Provider also tracks current position and provides it to SiteMapPath controls
- ASP.NET 2.0 ships with one provider
  - XmlSiteMapProvider
- Use custom providers for other data stores

SiteMap API

- System.Web.SiteMap represents site maps
  - RootNode property identifies root node
  - CurrentNode property identifies current node
- SiteMapNode represents nodes
  - Interrogate properties of node
  - Walk up, down, and sideways in the hierarchy
- The magic underlying SiteMapPath controls
- Foundation for building controls of your own

Using the SiteMap API

```
// Write the title of the current node to a Label control

Label1.Text = SiteMap.CurrentNode.Title;
// Write the path to the current node to a Label control
SiteMapNode node = SiteMap.CurrentNode;
StringBuilder builder = new StringBuilder (node.Title);
while (node.ParentNode != null) {
    node = node.ParentNode;
    builder.Insert (0, " > ");
    builder.Insert (0, node.Title);
}

Label1.Text = builder.ToString ();
```

The ImageMap Control

```
<asp:ImageMap ImageUrl="Shapes.jpg" OnClick="OnUpdate" RunAt="server">
  <asp:CircleHotSpot X="50" Y="50" Radius="50" PostBackValue="Circle"
    AlternateText="Circle" HotSpotMode="Postback" RunAt="server" />
  <asp:RectangleHotSpot Left="0" Top="100" Right="100" Bottom="200"
    PostBackValue="Rectangle" AlternateText="Rectangle"
    HotSpotMode="Postback" RunAt="server" />
  <asp:PolygonHotSpot Coordinates="50, 200, 0, 300, 100, 300"
    PostBackValue="Triangle" AlternateText="Triangle"
    HotSpotMode="Postback" RunAt="server" />
</asp:ImageMap>
  ...
<script language="C#" runat="server">
void UpdateLabel (Object sender, ImageMapEventArgs e)
{
  ...
}
</script>
```

The FileUpload Control

```
<asp:FileUpload ID="UploadControl" RunAt="server" />
<asp:Button Text="Upload" OnClick="OnUpload" RunAt="server" />
  .
  .
<script language="C#" runat="server">
void OnUpload (Object sender, EventArgs e)
{
    if (FileUploadControl.HasFile) {
        string name = UploadControl.PostedFile.FileName;     // Path name
        Stream bits = UploadControl.PostedFile.InputStream; // Contents
          ...
        // Use the SaveAs method to persist to a local file
        FileInfo file = new FileInfo (UploadControl.PostedFile.FileName);
        UploadControl.SaveAs (Server.MapPath ("~/Uploads/" + file.Name);
    }
}
</script>
```

The MultiView Control

```
<asp:MultiView ID="Main" ActiveViewIndex="0" RunAt="server">
  <asp:View RunAt="server">
    ...
  </asp:View>
  <asp:View RunAt="server">
    ...
  </asp:View>
  <asp:View RunAt="server">
    ...
  </asp:View>
</asp:MultiView>
  .
  .
void OnSwitchView (Object sender, EventArgs e)
{
    Main.ActiveViewIndex = 1; // Switch views
}
```

Declarative View Switching

```
<asp:MultiView ID="Main" ActiveViewIndex="0" RunAt="server">
  <asp:View RunAt="server">
    ...
    <asp:Button CommandName="SwitchViewByIndex" CommandArgument="1"
      Text="Switch to view 2" RunAt="server" />
  </asp:View>
  <asp:View RunAt="server">
    ...
    <asp:Button CommandName="SwitchViewByIndex" CommandArgument="0"
      Text="Switch to view 1" RunAt="server" />
  </asp:View>
</asp:MultiView>
```

IIS

Internet Information Server

- Microsoft's web server
- Foundation for ASP.NET
- Runs in inetinfo.exe process
- Also FTP, NNTP, SMTP
- Shared resources
  - Default location c:\inetpub\wwwroot
  - Internet Services Manager
- A Microsoft Management Console (MMC) snap-in

IIS Virtual Directories

Provides a level of indirection from URL to actual file locations on the server

For example, the file for the url: **http://myServer/TransflowerOnline/tfl.asp**
could be mapped to the physical location: **d:\myFolder\ TransflowerOnline\tfl.asp**

Web Application

What Is a Web Application?
All resources (files, pages, handlers, modules, executable code, etc.) within a virtual directory and its subdirectories

- Configuration files
- Shared data (application variables)
- global.asax
- Scopes user sessions

A session is a series of web page hits by a single user within a block of time
Shared data (session variables)

WebApplication global.asax

Located at application root
Can define and initialize application variables and session variables
Specify object to create with class, COM ProgID or COM ClassID
Be careful: use of shared objects can cause concurrency errors or blocking!

```
<object id="items" runat="server"
  scope="application"
  class="System.Collections.ArrayList" />
```

Can contain user-created code to handle application and session events (just like ASP)
Application_OnStart, Application_OnEnd
Session_OnStart, Session_OnEnd

```
void Application_OnStart () {
  Application ["startTime"]=DateTime.Now.ToString();
}
void Session_OnStart() {
  Session["startTime"]=DateTime.Now.ToString();
}
```

Can use code-behind
Can contain application-level directives

```
<%@ Application Description="This is my app..." %>
<%@ Application Inherits="MyBaseClass" %>
<%@ Application Src="Global.cs" %>
            Visual Studio.NET uses this
<%@ Import Namespace="System.Collections" %>
<%@ Assembly Name="MyAssembly.dll" %>
```

Configuration

Goal
Provide extensible configuration for admins & developers to hierarchically apply settings for an application

Solution
Store configuration data in XML text files
Format is readable and writable by people and machines
Settings specified in configuration sections, e.g.
Security, SessionState, Compilation, CustomErrors, ProcessModel, HTTPHandlers, Globalization, AppSettings, WebServices, WebControls, etc.
Configuration information stored in web.config
It is just a file, no DLL registration, no Registry settings, no Metabase settings
<!-- web.config can have comments -->

Configuration Hierarchy

Configuration files can be stored in application folders
Configuration system automatically detects changes
Hierarchical configuration architecture
Applies to the actual directory and all subdirectories

Configuration web.config Sample

```
<configuration>
 <configsections>
      <add names="httpmodules"
            type="System.Web.Config.HttpModulesConfigHandler"/>
      <add names="sessionState"
            type="..."/>
 </configsections>

 <httpModules>
 <!--- http module subelements go here -->
 </httpModules>
<sessionState>
    <!--- sessionstate subelements go here -->
  </sessionState>
</configuration>
```

Configuration Hierarchy

Standard machine-wide configuration file
Provides standard set of configuration section handlers
Is inherited by all Web Applications on the machine

C:\WINDOWS\Microsoft.NET\...\...\config\machine.config

Configuration User Defined Settings

Create web.config in appropriate folder

```
<configuration>
<appSettings>
 <add key="CxnString"
      value="localhost;uid=sa;pwd=;Database=tfl"/>
</appSettings>
</configuration>
```

Retrieve settings at run-time

```
string cxnStr = ConfigurationSettings.AppSettings["CxnString"];
```

Custom Configuration Handlers

Extend the set of section handlers with your own
Implement the interface:
`System.Web.Configuration.IConfigurationSectionHandler`
Add to web.config or machine.config

Tracing

ASP.NET supports tracing
- Easy way to include "debug" statements
  - No more messy Response.Write() calls!
  - Debug statements can be left in, but turned off
- Great way to collect request details
  - Server control tree
  - Server variables, headers, cookies
  - Form/Query string parameters
  - Tracing provides a wealth of information about the page
- Can be enabled at page- or application- level

Tracing Methods and Properties

Methods
- Trace.Write: Writes category and text to trace
- Trace.Warn: Writes category and text to trace in red

Properties
- Trace.IsEnabled: True if tracing is turned on for the application or just that page
- Trace.Mode: SortByTime, SortByCategory

Implemented in System.Web.TraceContext class

Page Level Tracing

To enable tracing for a single page:
- Add trace directive at top of page
  - <%@ Page Trace="True" %>
- Add trace calls throughout page
  - Trace.Write("MyApp", "Button Clicked");
  - Trace.Write("MyApp", "Value: " + value);
- Access page from browser

Application Level Tracing

To enable tracing across multiple pages:
Create web.config file in application root

```
<configuration>
<trace enabled="true" requestlimit="10"/>
</configuration>
```

Hit one or more pages in the application
Access tracing URL for the application

```
http://localhost/MyApp/Trace.axd
```

Tracing into a Component

To add tracing to a component:

Import the Web namespace: using System.Web;

Enable tracing in your class constructor (optional): HttpContext.Current.Trace.IsEnabled = true;

Write to trace: HttpContext.Current.Trace.Write("category","msg");

State Management The Problem

How/where to store data?
How can you pass data from page to page?
How do we get around HTTP statelessness?

State Management Three-Tier Architecture

State Management Client

Client-side state management means this:
- Client requests an initial page
  - The server generates a HTTP/HTML response that is sent back to the client
    - This response includes data (state)
  - User looks at the response and makes a selection, causing another request to the server
    - This second request contains the data that was sent in the first response
  - The server receives and processes the data
    - Could be same server or different server

URL in a hyperlink (<a>)
- Query string
- Very visible to users
  - This can be good or bad

Hidden form elements
- Like __VIEWSTATE

Cookies
- Limited to 4K
- May be blocked by users

State Management Web Server Middle-Tier

Application variables
- Shared by all sessions (users)

Session variables
- Still need to pass session id through the client
- ASP.NET State Service or database

Caching
Similar to application variables, but can be updated periodically or based upon dependencies

State Management Database

Application-level
Part of application database design
Session-level
Custom session state management in database
ASP.NET database session state support

State Management in ASP.NET
ASP.NET supports both Application-level and Session-level state management
Allows you to store state (data) in middle tier

State Management Application Variables
Application state is stored in an instance of HttpApplicationState
Accessed from Page.Application property
Can lock Application object for concurrent use
Needed only when changing application variable
Again, use this wisely
Use in "read-mostly" style
Initialize in global.asax
Avoid serializing your pages

State Management Sessions

What is a session?
Context in which a user communicates with a server over multiple HTTP requests
Within the scope of an ASP.NET Application
HTTP is a stateless, sessionless protocol
ASP.NET adds the concept of "session"
Session identifier: 120 bit ASCII string
Session events: Session_OnStart, Session_OnEnd
Session variables: store data across multiple requests
ASP.NET improves upon ASP sessions

State Management Session Identifier
By default, session id stored in a cookie
Can optionally track session id in URL
New in ASP.NET
Requires no code changes to app
All relative links continue to work

```
<configuration>
    <sessionState cookieless="true"/>
</configuration>
```

State Management Session Variables
ASP stores session state in IIS process
State is lost if IIS crashes
Can't use session state across machines
ASP.NET stores session state:

Prasanna Wadekar

In process
In another process: ASP State NT service
In SQL Server database

```
<sessionstate inproc="false"
      server="AnotherServer" port="42424" />

<sessionstate inproc="false"
      server="AnotherServer" port="42424"
           usesqlserver="true" />
```

"Live" objects are are not stored in session state
Instead, ASP.NET serializes objects out between requests
ASP.NET approach provides:
Ability to recover from application crashes
Ability to recover from IIS crash/restart
Can partition an application across multiple machines (called a Web Farm)
Can partition an application across multiple processes (called a Web Garden)

Transferring Controls between Pages
Link to a page
Postback
Response.Redirect
Causes HTTP redirect
Tells browser to go to another URL
Server.Transfer
Like a redirect but entirely on one server
Server.Execute
Execute one page from another then return control
Both pages processed on same server

Caching

Many sites spend considerable effort generating the same web pages over and over
For example, a product catalog changes overnight, but is accessed tens of thousands of times a day
Server-side caching can vastly improve performance and scalability
ASP.NET provides support for
Page output caching
Data caching

Page out Caching

Entire web page output (HTML) is cached
Must specify life of cached page (in seconds)
Can cache multiple versions of a page, by:
GET/POST parameters; use VaryByParam
HTTP header; use VaryByHeader
E.g. Accept-Language
Browser type or custom string; use VaryByCustom

Partial Page out Caching

Can cache a portion of a page by placing it in a User Control
Can cache multiple versions of a User Control on a page, by property; use VaryByControl

Caching in the Browser

Don't confuse server-side page output caching with how the browser and proxy servers cache the page
Use Response.Cache to specify HTTP cache headers
Contains a HttpCachePolicy object

Data Caching

Data cache is similar to application variables
Can cache objects, HTML fragments, etc.
Usage pattern:
Try to retrieve data
If null then create data and insert into cache

```
DataView Source = (DataView)Cache["MyData"];
if (Source == null) {
  Source = new DataView(ds.Tables["Authors"]);
  Cache["MyData"] = Source;        // Save in cache
}
```

Cache object is stored on the Page and is an instance of System.Web.Caching.Cache
Cache may be scavenged: when memory runs low it will be automatically reclaimed
Can specify data expiration: absolute time (e.g. midnight), relative (in 1 hour)
Cached data can be dependent upon a file or other cache item

```
Cache.Insert("MyData", Source, null, // Expire in 1 hour
  DateTime.Now.AddHours(1), TimeSpan.Zero);
Cache.Insert("MyData", Source,        // Dependent on file
  new CacheDependency(Server.MapPath("authors.xml")));
```

Populating a data cache has an inherent race condition: if hit almost concurrently, multiple pages may try to populate the same cache

This probably doesn't matter at all; it's only significant if the cost to create the data is prohibitive or if there are side effects

If it does matter, there are two solutions:

- Populate the cache in Application_OnStart
- Synchronize access to the cache

```
private static String cacheSynchronize = "tflKey";;
DataView Source = (DataView)Cache["tflDataSet"];
if (Source == null) {
  lock (cacheSynchronize) {
    Source = (DataView)Cache["tflDataSet"];
    if (Source == null) {        // Have to test again
      // Open database ...
      Source = new DataView(ds.Tables["Trainers"]);
      Cache["MyDataSet"] = Source;    // Save in cache
    }
  }
}
```

ASP.NET page state maintenance is great, but __VIEWSTATE can get quite large

Why store constant data in __VIEWSTATE?

E.g. dropdowns listing countries, states, days of the week, months, product categories, SKUs, etc.

Instead, set EnableViewState=false, cache that data on the server, and populate the control from the cache in Page_Load

- Can cache data or even HTML
  - Use Control.Render() to obtain a control's HTML

Error Handling

.NET Common Language Runtime provides a unified exception architecture

- Runtime errors done using exceptions
- VB now supports try/catch/finally

ASP.NET also provides declarative application custom error handling

- Automatically redirect users to error page when unhandled exceptions occur
- Prevents ugly error messages from being sent to users

Error Handling Custom Error pages

- Can specify error pages for specific HTTP status codes in web.config

```
<configuration>
  <customerrors mode="remoteonly"
                defaultredirect="error.htm">
    <error statuscode="404"
           redirect="adminmessage.htm"/>
    <error statuscode="403"
           redirect="noaccessallowed.htm"/>
  </customerrors>
</configuration>
```

Error Handling Error Events

Can override Page.HandleError to deal with any unhandled exception on that page
Global application event raised if unhandled exception occurs
- Provides access to current Request
- Provides access to Exception object
- See HttpApplication.Error event

What do you actually do when an error occurs?
- Use new EventLog class to write custom events to log when errors occur
- Use new SmtpMail class to send email to administrators

Error Handling writing to Event Log

```
<%@ Import Namespace="System.Diagnostics" %>
<%@ Assembly name="System.Diagnostics" %>
<script language="C#" runat=server>
public void Application_Error(object Sender, EventArgs E) {
  string LogName = "MyCustomAppLog";
  string Message = "Url " + Request.Path + " Error: " +
                   this.Error.ToString()
  // Create event log if it doesn't exist
  if (! EventLog.SourceExists(LogName)) {
    EventLog.CreateEventSource(LogName, LogName);
  }
  // Fire off to event log
  EventLog Log = new EventLog();
  Log.Source = LogName;
  Log.WriteEntry(Message, EventLogEntryType.Error);
}
</script>
```

Error Handling Sending SMTP Mail

```
<%@ Import Namespace="System.Web.Util" %>
<%@ Assembly name="System.Diagnostics" %>
<script language="C#" runat=server>
public void Application_Error(object Sender, EventArgs E) {
  MailMessage MyMessage = new MailMessage();
  MyMessage.To = "ravi.tambade@transflower.in";
  MyMessage.From = "TFLAppServer";
  MyMessage.Subject = "Unhandled Error!!!";
  MyMessage.BodyFormat = MailFormat.Html;
  MyMessage.Body = "<html><body><h1>" + Request.Path +
    "</h1>" + Me.Error.ToString() + "</body></html>";
  SmtpMail.Send(MyMessage);
}
</script>
```

Personalization and Localization is ASP.NET

Profile Service

Stores per-user data persistently
- Strongly typed access (unlike session state)
- On-demand lookup (unlike session state)
- Long-lived (unlike session state)
- Supports authenticated and anonymous users

Accessed through dynamically compiled HttpProfileBase derivatives (HttpProfile)
Provider-based for flexible data storage

Profile Schema

Defining a Profile

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="ScreenName" />
        <add name="Posts" type="System.Int32" defaultValue="0" />
        <add name="LastPost" type="System.DateTime" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Using a Profile

```
// Increment the current user's post count
Profile.Posts = Profile.Posts + 1;
// Update the current user's last post date
Profile.LastPost = DateTime.Now;
```

How Profile Works

Auto generated class representing the page

```
public partial class page_aspx : System.Web.UI.Page
{
  ...
    protected ASP.HttpProfile Profile
    {
        get { return
((ASP.HttpProfile)(this.Context.Profile)); }
    }
  ...
}
```

Auto generated class derived from *httpProfileBase*

Profile property included in auto generated page class

Accessing another User's Profile

Profile.propertyname refers to current user

Use Profile.GetProfile (username) to access profiles for other users

```
// Get a reference to Ravi's profile
HttpProfile profile = Profile.GetProfile ("Ravi");
// Increment Fred's post count
profile.Posts = profile.Posts + 1;
// Update Ravi's last post date
profile.LastPost = DateTime.Now;
```

Accessing a profile from External Component

"Profile" property is only valid in classes generated by ASP.NET (ASPX, ASAX, etc.)

Use HttpContext.Profile property to access profiles from external components

```
// Read the current user's ScreenName property in an ASPX file
string name = Profile.ScreenName;
// Read the current user's ScreenName property in an external component
string name = (string) HttpContext.Current.Profile["ScreenName"];
```

Profile Groups

Properties can be grouped

<group> element defines groups

```
<profile>
  <properties>
    <add ... />
    <group name="...">
        <add ... />
    </group>
  </properties>
</profile>
```

Defining a Profile Group

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="ScreenName" />
        <group name="Forums">
          <add name="Posts" type="System.Int32" defaultValue="0" />
          <add name="LastPost" type="System.DateTime" />
        </group>
      </properties>
    </profile>
  </system.web>
</configuration>
```

Accessing a Profile Group

```
// Increment the current user's post count
Profile.Forums.Posts = Profile.Forums.Posts + 1;

// Update the current user's last post date
Profile.Forums.LastPost = DateTime.Now;
```

Custom DataType

Profiles support base types
- String, Int32, Int64, DateTime, Decimal, etc.

Profiles also support custom types
- Use type attribute to specify type
- Use serializeAs attribute to specify serialization mode: Binary, Xml (default), or String

serializeAs="Binary" types must be serializable
serializeAs="String" types need type converters

Using a Custom DataType

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="Cart" type="ShoppingCart" serializeAs="Binary" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Anonymous User Profiles

By default, profiles aren't available for anonymous (unauthenticated) users
- Data keyed by authenticated user IDs

Anonymous profiles can be enabled
- Step 1: Enable anonymous identification
- Step 2: Specify which profile properties are available to anonymous users

Data keyed by user anonymous IDs

Profiles for Anonymous Users

```
<configuration>
  <system.web>
    <anonymousIdentification enabled="true" />
    <profile>
      <properties>
        <add name="ScreenName" allowAnonymous="true" />
        <add name="Posts" type="System.Int32" defaultValue="0 />
        <add name="LastPost" type="System.DateTime" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Anonymous Identification

Anonymous identification can be cookied or cookieless (URL munging)
Cookies can be configured

```
<anonymousIdentification
      enabled="[true|false]"
      cookieName=".ASPXANONYMOUS"
       cookieTimeout="100000"
      cookiePath="/"
      cookieRequireSSL="[true|false]"
       cookieSlidingExpiration="[true|false]"
       cookieProtection="[None|Validation|Encryption|All]"
      cookieless="[UseCookies|UseUri|AutoDetect|UseDeviceProfile]"
/>
```

Profile Events

Profile service and anonymous identification service fire global events

Global.asax Handler Name	Description
Anonymousidentification_Create	Called when anonymous ID is issued
Anonymousidentification_Remove	Called when authentication request contains anonymous ID and ID is therefore deleted
Profile_MigrateAnonymous	Called other anonymousidentification_Remove to migrate settings for anonymous users

Profile Providers

Profile service is provider-based
VS 2005 ships with SqlProfileProvider (SQL Server)
Use custom providers to add support for other data stores

Localization

@ Page (UI)Culture "auto" keyword
Declarative mapping of Accept-Language headers to relevant thread properties
Simplified resource handling
Declarative mapping of control properties to resources using <%$ ... %> expressions
Strongly typed programmatic resource loading
<localize runat="server"> and more

Culture Handling (ASP.NET 1.x)

Code required to map Accept-Language headers to Current (UI) Culture properties of current thread

Global.asax

```
<script language="C#" runat="server">
void Application_BeginRequest (Object sender, EventArgs e)
{
    Thread.CurrentThread.CurrentCulture =
        CultureInfo.CreateSpecificCulture (Request.UserLanguages[0]);
    Thread.CurrentThread.CurrentUICulture =
        Thread.CurrentThread.CurrentCulture;
}
</script>
```

Culture Handling (ASP.NET)
"auto" keyword maps Accept-Language headers to Current(UI)Culture properties of current thread
Single Page (ASPX)

```
<%@ Page Culture="auto" UICulture="auto" %>
```

All pages (Web.config)

```
<globalization culture="auto" uiCulture="auto" />
```

Localization Resources

Application resources
Available to all pages in application
RESX files in ~/App_GlobalResources subdirectory
Local resources
Available to a specific page
RESX files in App_LocalResources subdirectories
Use filename.culture.resx naming schema for localization based on CurrentUICulture

Application Resources

Loading Application Resources

Declarative

```
<asp:Label ID="Output" RunAt="server"
        Text='<%$ Resources:AppResources, Greeting %>' />
```

(RESX Name: AppResources) (Resource Name: Greeting)

Programmatic

```
// Strong typing
Output.Text = Resources.AppResources.Greeting;
// Weak typing
Output.Text = (string) GetAppResourceObject ("AppResources", "Greeting");
```

Loading Local Resources

Declarative

```
<asp:Label ID="Output" RunAt="server" Text='<%$ Resources:Greeting%>' />
```

Resource Name

Programmatic

```
// Weak typing only
Output.Text = (string) GetPageResourceObject ("Greeting");
```

Implicit Expressions

Batch-initialize control properties
Applicable to local resources only

foo.aspx

```
<asp:Label ID="Output" RunAt="server" meta:ResourceKey="MyLabel" />
```

The Localize Control

Loads static content from resources based on Thread.CurrentThread.CurrentUICulture
foo.aspx

```
<asp:Localize Runat="server" Text="<%$ Resources:WelcomeMessage %>" />
```

Panel. Direction

Direction="RightToLeft" right-justifies content generated by child controls
Use it to right-justify right-to-left text
Supported at design time by VS IDE

```
<asp:Panel Direction="RightToLeft" RunAt="server">
     Imagine this were in Hebrew<br />
   <asp:Login RunAt="server" />
</asp:Panel>
```

Deployment

- XCOPY deployment
  - Components are placed in .\bin folder
- No DLL deployment, registration
  - Unless you're using COM or other DLLs
- No locked DLLs
  - DLLs are "shadow copied" into a hidden folder
  - .aspx files are automatically compiled
  - Not true for codebehind
- Update code (.aspx and assemblies) while server is running
- No need to stop/bounce the server

- Applications are isolated
  - Each can have their own version of components
- Uninstall = delete /s *.*

Security

- Reasons for Security
  - Prevent access to areas of your Web server
  - Record and store secure relevant user data
- Security Configuration
  - <security> tag in web.config file
- Authentication, Authorization, Impersonation
- Code Access Security
  - Are you the code you told me you are?
  - Protect your server from bad code

Security Authentication

- Who are you?
- Server must authenticate client
- Client should authenticate server
  - Kerberos does
- Need a directory to store user accounts
  - NT: Security Accounts Manager
    - Good for intranet usage up to about 40,000 accounts
  - Windows Server: Active Directory
    - Good for intranet and Internet usage

Security IIS Authentication

- Anonymous
  - A single Windows Server account is used for all visitors
- Basic authentication
  - Standard, commonly supported
  - Password sent in clear text
- Digest authentication
  - Standard, but not yet common
- Integrated Windows Authentication
  - NTLM
  - Kerberos (Windows 2000 only)
- Client certificates
  - Mapped to W2K/NT account

Security ASP.NET Authentication

- Passport module provided
  - Exposes passport profile API
- Custom, forms-based authentication
  - Easy to use, with cookie token tracking
  - Enables custom login screen (no popup dialogs)
  - Supports custom credential checks against database, exchange, etc.

Security Authorization

- Now that I know who you are, here's what you are allowed to do
- W2K/NT DACLs (Discretionary Access-Control List)
  - Grant and deny read/write/execute/etc. permission to users or groups of users
- IIS also provides coarse-grained control
  - Read, write, run script, run executable, directory browsing, script access for virtual directories, directories and files

Security ASP.NET Authorization

ASP.NET supports authorization using either users or roles
Roles map users into logical groups
Example: "User", "Manager", "VP", etc.
Provides nice developer/admin separation
Developers can perform runtime role checks in code
if (User.IsInRole("Admin") { }

Security Impersonation

IIS authenticates the "user"
A token is passed to the ASP.NET application
ASP.NET impersonates the given token
Access is permitted according to NTFS settings

Developing Web Forms Debugging

- Basic Debugger ships with the SDK
  - Multi-language
  - Single stack trace
  - Breaks, watches, etc.
- Visual Studio® Debugger
  - Adds remote debugging support
  - Supports managed/unmanaged debugging

1. Create web.config file in application root

```
<configuration>
<compilation debugmode="true"/>
</configuration>
```

2. Attach using debugger
3. Set breakpoints
4. Hit page or service

ASP.NET MVC

ASP.NET
: One Web application framework to rule them all...

Caching	Modules	Globalization
Pages	Controls	Master Pages
Profile	Roles	Membership
Handlers	Etc.	

ASP.NET MVC
Separation of concerns
- Model
- View
- Controller

How does MVC Look?

There are two major considerations for choosing between Webforms and ASP.NET MVC:

- **Test Driven Development** – life would be MUCH easier using MVC while following TDD.
- **Data Driven Application** – life would be MUCH easier using WebForms if the application is data heavy.

There are no rights or wrongs, and every application can be written in both frameworks. In fact, you can even have a hybrid approach, where you can write some part in WebForms and some in MVC.

SWOT Analysis Webforms and ASP.NET MVC

Asp.Net WebForms

Strengths

- Provides very good RAD development capabilities.
- Great designer support in Visual Studio.
- Ease of development for data-heavy LOB applications.
- Very rich control libraries and third party vendor support.
- A familiar event-driven model when compared to Windows Forms development, and so easy for developers to pick up.

Weaknesses

- UI logic coupled with the code, and thus is hard to separate.
- Harder to unit test, so difficult to employ TDD.
- Heavy page sizes due to view state management.

Opportunities

- Great at creating quick prototypes for business applications. This comes in very handy when trying to show quick Proof of Concepts to clients.

Threats

- Harder to adopt to various UI views despite the various frameworks available (master pages, Themes, etc.).

WebForms is great, but options are good.....

Classic ASP.NET (Web Forms)

High level abstraction over HTML / HTTP

Simplified state management

ViewState and the post-back model

Control model

Data binding

Simple event-driven mechanism

Simple Page Controller pattern

What makes it lazy ..... web forms

Sub-optimal URLs

blog.aspx?date=21032008

Form runat="server"

ViewState

Hard to test

All sorts of code in the page

Requirement to test with an HttpContext

No real role responsibility...

Pages, Master pages, UI logic, Business logic, Data Access everything together with the help of powerful Webserver controls.

Control abstractions can be negative...

<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJMzQyMDk1NTAPZBYCZg9kFgICAw9kFgQCAw9kFgYCAQ8WAh4Hcml2aWJsZ
WgWAgIDDw8WAh4LTmF2aWdhQVXcmwFKC9TdG9yZS90b2Nob21lcHNlcnNpc2Uvx4NlcEFjY
29lcnR2LmF2cHNkZAIDDxYCHwEoF
QICAwSPFgIEFA1eWFZGQCEQ9kFgJmD2QWEQIBDw8WAh8BBVZodHRwcZovL3dkdy5jb25zd
WlcmWYvZDcy5jb20vUsRvcmuvTGVnaW4uYXNwaG9SZXRlcmVvcmV0L1N0b3JlLOR1cmF1b
HQuYXNwZCYyZWc6VHJ12WkFAgUPD
xYCHwEFNwh0dHA8Ly93d7cuY29u2lvtZvJlcmFndH4uY26tL1N0b3JlL1N0b28wlW5nQ2Fyd
C5hc3B4ZGQCHw8PFgIfAQVJaHR0cHcvu3d3dy5jb25zdWlcmHyYWZCcy5jb25vU3Rvcmuvv
NnlcmVmHZb3jpdGvZLmF2CHhkI
AIJDw8WAIHB6THodHRwO18vd3d3LmNvbnN1bWVyY3JhZnR2LmNvb59TdG9yZS9vc2VycY4Ne
UFjY29LbnQuYXNwZGRkAgsFDxYCHwIFATBkZAIKDw8WAh8BBTVodHRwO18vd3d3LmNvbnN1b
WVyY3JhZnR2LmNvb59TdG9yZS9Ta
C9wcGluZONlcnQuYXNwZGRkAg8PDxYCHwIFBDQwLjAwZGQCEQ8PFgQfAQUlaHR0cDovL3d3d
y5jb25zdWlcmHyYWZCcy5jb2OvU3RvcmUvU2hvcHBpbmQtYX3GLmF2cHgcEBUVuFWJSZWR0Z
GQCE9kFgRmD2QWAgIED1YCHgVlb
GFzcwUPdG9wX2NhcnRfb3JhbmdlFqnCAQ8WAh8EBRZ0b3Bfy2FydF9vcmFkZ2VfaGVhZGVyF
gICAwRkFgICAQ8WAh8CBAJTYY2I2CBObyBmYXZvcml0ZXNkAgMPZBYIZg9kFgJmD2QWAgIBD
w8WAh8CZGRkAgEPZBYCZg9kFgQCA
Q8WAh8CZGQCBQ8WAh8CZGQCAg8WAh8A4bYEZg9kFgICAwBPFgIfAgUEMZ8kAgEPZBYCAgEPD
xYCHwIFBSQwLjAwZGQCAw8WAh8AaBYEZg9kFgICAQ8PFgIfAgUBMzRkAgEPZBYCAgEPDxYCH
wIFBSQwLjAwZGQCBQ8PFgIfCFlrY
wdlvXJsZGRkAgCPFgIfBAULYmVhZF9jb3Z1cjqWAgIBDxYCHgRoGnVmZGQCCQ8WAh4PU3R5b
GUFDWRpc3BsYXQ6bm0uZTsWAgIBDw8WAh8DaGRkAgTPZBYCAgMPFgQeDk9uQ2FlY2V6U2Hya
XB0BXFTZXRQb3B1cFZpc21lawxpd
HkoJ2W0bDAwV2hlYwRlcHVnRzyb2xfaxRlbuFkZGVkVG90YXJ0UG9wdxBEazZnWGSF5xC5vb
kIVZGFsUG9wdxBOb25GcmQxG12vmlzawJsZUZpZwvkJywgZmFsc2UpNx4KT25Pa1hjcmlwd
AvxU2V0UG9wdxBWaXHgYmlsaxR5k

It isn't easy enough to test...

Asp.Net MVC

Strengths

- Provides fine control over rendered HTML.
- Cleaner generated HTML.
- Superior separation between UI and code.
- Easier to unit test.
- Can support multiple view engines.
- By default uses RESTful interfaces for URLs – so better SEO.
- No ViewState (this may also be a weakness).
- Typical size of page is small.
- Easy integration with frameworks like JQuery.

Weaknesses

- Not event driven, so maybe difficult for people who know only Asp.Net Webforms to wrap their minds around it.
- Third party control library support is not that strong.
- No ViewState(this is also a strength).

Opportunities

- Allows for Test Driven Development (TDD) – it is built with TDD in mind, so its much easier to write unit test cases, mock objects, and to intercept the program flow.
- Allows for reuse of the same models to present different UIs and Interfaces.

Threats

- Bigger ramp-up and training time required for developers with no or little experience in web application development.

ASP.NET MVC Tenets

Alternative
Testable
Extensible
Routable

Choosing between ASP.NET Webforms and ASP.NET MVC

ASP.NET MVC doesn't have...

Postbacks
View state
Control state
Server-side form
Page/Control lifecycle

ASP.NET MVC still has...

Web designer
Master pages & User controls
Membership/Roles/Profile
Globalization & Caching
HTTP intrinsics:
- HttpContext
- HttpRequest
- HttpResponse
- Etc.

Routing

Routing provides "clean" URLs
URL is mapped to a route handler
Extra level of indirection
Handlers can be changed without impacting URL
URL can be changed without impacting handler
Enables support for multilingual URLs
URL Example: http://www.mysite.com/Home/ProductList

Developers adds Routes to a global RouteTable
Mapping creates a RouteData - a bag of key/values

```
public static void RegisterRoutes(RouteCollection routes)
{
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
routes.MapRoute( "Default",
                "{controller}/{action}/{id}",
            new {  controller = "Home", // Parameter defaults
            action = "Index",
                  action = "Index",  id = ""
}
};
}
protected void Application_Start()
{
      RegisterRoutes(RouteTable.Routes);
}
```

MVC Controllers

Scenarios, Goals and Design
URLs route to controller "actions",
not pages – mark actions in Controller.
Controller executes logic, chooses view.
All public methods are accessible

```
public void ShowPost(int id) {
   Post p = PostRepository.GetPostById(id);
   if (p != null) {
       RenderView("showpost", p);
   } else {
       RenderView("nosuchpost", id);
   }
}
```

Controller Conventions

Controller
Must...
Be suffixed with "Controller"
Implement IController (or inherit from Controller)
Action
Must...
Be Public
Return ActionResult or void
Can't...
Be generic
Have a NonActionAttribute
Have out/ref parameters

Views

- Are for rendering/output.
- Pre-defined and extensible rendering helpers
- Can use .ASPX, .ASCX, .MASTER, etc.
- Can replace with other view technologies
- Template engines (NVelocity, Brail, ...).
- Output formats (images, RSS, JSON, ...).
- Mock out for testing.
- Controller sets data on the View
- Loosely typed or strongly typed data

Clean URLs
Don't settle for...
/Products.aspx?CategoryID=123
When you can easily have...
/Product/Puppies

Visual Studio

ASP.NET MVC Features

- Razor View Engine
- Multiple View Engine Support
- Validation Improvements
- Dynamic ViewBag
- Global Filters
- Action Results
- Taskbased Helpers
- Dependency Injection
- Porting MVC Libraries to jQuery
- Granular Validation Input
- Caching Support
- JSON Binding Support

```
protected void Application_Start()
{
    ViewEngines.Engines.Add(new SparkViewFactory());
    ...
}
```

Razor

The Razor View Engine
Razor syntax is clean and concise, requiring a minimum number of keystrokes.
Razor is easy to learn, in part because it's based on existing languages like C# and Visual Basic.
Visual Studio includes IntelliSense and code colorization for Razor syntax.
Razor views can be unit tested without requiring that you run the application or launch a web server.

Syntax Comparison
Using traditional asp.net code

```
<h1>Code Nugget Example with .ASPX</h1>
```

```
<h3>Hello <%=name %>, the year is <%= DateTime.Now.Year %></h3>
<p>Checkout <a href="/Products/Details/<%=productId %">this product</a>
</p>
```

Using Razor

```
<h1>Razor Example</h1>
<h3>
     Hello @name, the year is @DateTime.Now.Year
</h3>
<p>
     Checkout <a href="="/Products/Details/@productId">this product</a>
</p>
```

Filters

Filters are custom classes that provide both a declarative and programmatic means to add pre-action and post-action behavior to controller action methods

- Authorization filters
- Action filters
- Result filters
- Exception filters

Dynamic View Bag

MVC 2 controllers support a ViewData property that enables you to pass data to a view template using a late-bound dictionary API.

In MVC 3, you can also use somewhat simpler syntax with the ViewBag property to accomplish the same purpose.

For example, instead of writing ViewData["Message"]="text", you can write ViewBag.Message="text".

You do not need to define any strongly-typed classes to use the ViewBagproperty.

Dynamic property, you can instead just get or set properties and it will resolve them dynamically at run time.

```
<head>
<title>@ViewBag.Title<title>
</head>
```

Extensibility

Any of these can be replaces.

Dependency Injection Improvements

ASP.NET MVC provides better support for applying Dependency Injection (DI) and for integrating with Dependency Injection or Inversion of Control (IOC) containers. Support for DI has been added in the following areas:

Controllers (registering and injecting controller factories, injecting controllers).
Views (registering and injecting view engines, injecting dependencies into view pages).
Action filters (locating and injecting filters).
Model binders (registering and injecting).
Model validation providers (registering and injecting).
Model metadata providers (registering and injecting).
Value providers (registering and injecting).

Model Binding

Model binding is the process of creating .NET objects using the data sent by the browser in an HTTP request.
Default model binder search in following location and order for named parameters data.

Name	Description
Request.Form	Content HTML form element data
RouteData.Values	Application routes values
Request.QueryString	Data in the query string of the request URL
Request.Files	Files that have been uploaded as part of the request

Bind attribute: To include or exclude model properties from the binding process.
To include first and last name in person object. Person FirstName and LastName property value will be consider other will ignored.

```
public ActionResult Register([Bind(Include = "FirstName, LastName")]
                                    Person  person)
```

Example: To exclude sex property in person object.
Person sex property value will be ignored.

```
public ActionResult Register([Bind(Exclude = "Sex")] Person person)
```

MVC Interview Questions/Answers

What are the 3 main components of an ASP.NET MVC application?
1. M - Model
2. V - View
3. C - Controller

In which assembly is the MVC framework defined?
System.Web.Mvc

Is it possible to combine ASP.NET webforms and ASP.MVC and develop a single web application?
Yes, it is possible to combine ASP.NET webforms and ASP.MVC and develop a single web application.

What does Model, View and Controller represent in an MVC application?
Model: Model represents the application data domain. In short the applications business logic is contained with in the model.

View: Views represent the user interface, with which the end users interact. In short the all the user interface logic is contained with in the UI.

Controller: Controller is the component that responds to user actions. Based on the user actions, the respective controller, work with the model, and selects a view to render that displays the user interface. The user input logic is contained with in the controller.

What is the greatest advantage of using asp.net mvc over asp.net webforms?
It is difficult to unit test UI with webforms, where views in mvc can be very easily unit tested.

Which approach provides better support for test driven development - ASP.NET MVC or ASP.NET Webforms?
ASP.NET MVC

What are the advantages of ASP.NET MVC?
1. Extensive support for TDD. With asp.net MVC, views can also be very easily unit tested.
2. Complex applications can be easily managed
3. Seperation of concerns. Different aspects of the application can be divided into Model, View and Controller.
4. ASP.NET MVC views are light weight, as they donot use viewstate.

Is it possible to unit test an MVC application without running the controllers in an ASP.NET process?
Yes, all the features in an asp.net MVC application are interface based and hence mocking is much easier. So, we don't have to run the controllers in an ASP.NET process for unit testing.

Is it possible to share a view across multiple controllers?
Yes, put the view into the shared folder. This will automatically make the view available across multiple controllers.

What is the role of a controller in an MVC application?
The controller responds to user interactions, with the application, by selecting the action method to execute and alse selecting the view to render.

Where are the routing rules defined in an asp.net MVC application?
In Application_Start event in Global.asax

Name a few different return types of a controller action method?
The following are just a few return types of a controller action method. In general an action method can return an instance of a any class that derives from ActionResult class.
1. ViewResult
2. JavaScriptResult
3. RedirectResult
4. ContentResult
5. JsonResult

What is the significance of NonActionAttribute?
In general, all public methods of a controller class are treated as action methods. If you want prevent this default behaviour, just decorate the public method with NonActionAttribute.

What is the significance of ASP.NET routing?
ASP.NET MVC uses ASP.NET routing, to map incoming browser requests to controller action methods. ASP.NET Routing makes use of route table. Route table is created when your web application first starts. The route table is present in the Global.asax file.

What are the 3 segments of the default route, that is present in an ASP.NET MVC application?
1st Segment - Controller Name
2nd Segment - Action Method Name
3rd Segment - Parameter that is passed to the action method

Example: http://pragimtech.com/Customer/Details/5
Controller Name = Customer
Action Method Name = Details
Parameter Id = 5

ASP.NET MVC application, makes use of settings at 2 places for routing to work correctly. What are these 2 places?
1. Web.Config File : ASP.NET routing has to be enabled here.
2. Global.asax File : The Route table is created in the application Start event handler, of the Global.asax file.

What is the adavantage of using ASP.NET routing?
In an ASP.NET web application that does not make use of routing, an incoming browser request should map to a physical file. If the file does not exist, we get page not found error.

An ASP.NET web application that does make use of routing, makes use of URLs that do not have to map to specific files in a Web site. Because the URL does not have to map to a file, you can use URLs that are descriptive of the user's action and therefore are more easily understood by users.

What are the 3 things that are needed to specify a route?
1. URL Pattern - You can include placeholders in a URL pattern so that variable data can be passed to the request handler without requiring a query string.
2. Handler - The handler can be a physical file such as an .aspx file or a controller class.
3. Name for the Route - Name is optional.

Is the following route definition a valid route definition?
{controller}{action}/{id}

No, the above definition is not a valid route definition, because there is no literal value or delimiter between the placeholders. Therefore, routing cannot determine where to separate the value for the controller placeholder from the value for the action placeholder.

What is the use of the following default route?
{resource}.axd/{*pathInfo}
This route definition, prevent requests for the Web resource files such as WebResource.axd or ScriptResource.axd from being passed to a controller.

What is the difference between adding routes, to a webforms application and to an mvc application?
To add routes to a webforms application, we use MapPageRoute() method of the RouteCollection class, where as to add routes to an MVC application we use MapRoute() method.

How do you handle variable number of segments in a route definition?
Use a route with a catch-all parameter. An example is shown below. * is referred to as catch-all parameter.
controller/{action}/{*parametervalues}

What are the 2 ways of adding constraints to a route?
1. Use regular expressions
2. Use an object that implements IRouteConstraint interface

Give 2 examples for scenarios when routing is not applied?
1. A Physical File is Found that Matches the URL Pattern - This default behaviour can be overriden by setting the RouteExistingFiles property of the RouteCollection object to true.
2. Routing Is Explicitly Disabled for a URL Pattern - Use the RouteCollection.Ignore() method to prevent routing from handling certain requests.

What is the use of action filters in an MVC application?
Action Filters allow us to add pre-action and post-action behavior to controller action methods.

If I have multiple filters impleted, what is the order in which these filters get executed?
1. Authorization filters
2. Action filters
3. Response filters
4. Exception filters

What are the different types of filters, in an asp.net mvc application?
1. Authorization filters
2. Action filters
3. Result filters
4. Exception filters

Give an example for Authorization filters in an asp.net mvc application?
1. RequireHttpsAttribute
2. AuthorizeAttribute

Which filter executes first in an asp.net mvc application?
Authorization filter

ASP.NET MVC

What are the levels at which filters can be applied in an asp.net mvc application?
1. Action Method
2. Controller
3. Application

Is it possible to create a custom filter?
Yes

What filters are executed in the end?
Exception Filters

Is it possible to cancel filter execution?
Yes

What type of filter does OutputCacheAttribute class represents?
Result Filter

What are the 2 popular asp.net mvc view engines?
1. Razor
2. .aspx

What symbol would you use to denote, the start of a code block in razor views?
@

What symbol would you use to denote, the start of a code block in aspx views?
<%= %>

In razor syntax, what is the escape sequence character for @ symbol?
The escape sequence character for @ symbol, is another @ symbol

When using razor views, do you have to take any special steps to proctect your asp.net mvc application from cross site scripting (XSS) attacks?
No, by default content emitted using a @ block is automatically HTML encoded to protect from cross site scripting (XSS) attacks.

When using aspx view engine, to have a consistent look and feel, across all pages of the application, we can make use of asp.net master pages. What is asp.net master pages equivalent, when using razor views?
To have a consistent look and feel when using razor views, we can make use of layout pages. Layout pages, reside in the shared folder, and are named as _Layout.cshtml

What are sections?
Layout pages, can define sections, which can then be overriden by specific views making use of the layout. Defining and overriding sections is optional.

What are the file extensions for razor views?
1. .cshtml - If the programming lanugaue is C#
2. .vbhtml - If the programming lanugaue is VB

How do you specify comments using razor syntax?
Razor syntax makes use of @* to indicate the begining of a comment and *@ to indicate the end. An example is shown

below.

@* This is a Comment *@

LINQ Language Integrated Query

The C# 3.0 language enhancements build on C# 2.0 to increase developer productivity: they make written code more concise and make working with data as easy as working with objects. These features provide the foundation for the LINQ project, a general purpose declarative query facility that can be applied to in-memory collections and data stored in external sources such as XML files and relational databases.

The C# 3.0 language enhancements consist of:

- **Auto-implemented properties,** which automate the process of creating properties with trivial implementations.
- **Implicitly typed local variables,** which permit the type of local variables to be inferred from the expressions used to initialize them.
- **Implicitly typed arrays**, a form of array creation and initialization that infers the element type of the array from an array initializer.
- **Extension methods,** which make it possible to extend existing types and constructed types with additional methods.
- **Lambda expressions,** an evolution of anonymous methods that concisely improves type inference and conversions to both delegate types and expression trees.
- **Expression trees**, which permit lambda expressions to be represented as data (expression trees) instead of as code (delegates).
- **Object and collection initializers, which you can use** to conveniently specify values for one or more fields or properties for a newly created object, combining creation and initialization into a single step.
- **Query expressions**, which provide a language-integrated syntax for queries that is similar to relational and hierarchical query languages such as SQL and XQuery.
- **Anonymous types**, which are tuple types automatically inferred and created from object initializers.

Use of Automatically Implemented Properties
Easy Initialization with Object and Collection Initializers

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Customer c = new Customer(1);
            c.Name = "Maria Anders";
            c.City = "Berlin";

            Console.WriteLine(c);
        }

    }
}
```

Implicitly Typed Local Variables and Implicitly Typed Arrays

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {

        static void Main(string[] args)
        {
            List<Customer> customers = CreateCustomers();

            Console.WriteLine("Customers:\n");
            foreach (Customer c in customers)
                Console.WriteLine(c);
        }

        static List<Customer> CreateCustomers()
        {
           return new List<Customer>
           {
        new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
        new Customer(2) { Name = "Laurence Lebihan",  City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",   City = "London"    },
        new Customer(4) { Name = "Ann Devon",         City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",     City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",       City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",    City = "London"    },
        new Customer(8) { Name = "Liz Nixon",         City = "Portland"  }
           };
        }
    }
}
```

Extending Types with Extension Methods

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Customer> customers = CreateCustomers();

            Console.WriteLine("Customers:\n");
            foreach (Customer c in customers)
                Console.WriteLine(c);
        }

        static List<Customer> CreateCustomers()
        {
            return new List<Customer>
            {
        new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
        new Customer(2) { Name = "Laurence Lebihan",  City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",   City = "London"    },
        new Customer(4) { Name = "Ann Devon",         City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",     City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",       City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",    City = "London"    },
        new Customer(8) { Name = "Liz Nixon",         City = "Portland"  }
            };
        }
    }
}
```

Working with Lambda Expressions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
```

```
class Program
{
    static void Main(string[] args)
    {
        var customers = CreateCustomers();

        var addedCustomers = new List<Customer>
        {
            new Customer(9)    { Name = "Paolo Accorti", City = "Torino" },
            new Customer(10)   { Name = "Diego Roel", City = "Madrid" }
        };

        var updatedCustomers = customers.Append(addedCustomers);

        var newCustomer = new Customer(10)
        {
            Name = "Diego Roel",
            City = "Madrid"
        };

        foreach (var c in updatedCustomers)
        {
            if (newCustomer.Compare(c))
            {
             Console.WriteLine("The new customer was already in the list");
                return;
            }
        }
        Console.WriteLine("The new customer was not in the list");
    }

    static List<Customer> CreateCustomers()
    {
        return new List<Customer>
        {
        new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
        new Customer(2) { Name = "Laurence Lebihan",  City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",   City = "London"    },
        new Customer(4) { Name = "Ann Devon",         City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",     City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",       City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",    City = "London"    },
        new Customer(8) { Name = "Liz Nixon",         City = "Portland"  }
        };
    }
  }
}
```

Using Lambda Expressions to Create Expression Trees

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
```

```
class Program
    {
        static void Main(string[] args)
        {
            var customers = CreateCustomers();

            foreach (var c in FindCustomersByCity(customers, "London"))
                Console.WriteLine(c);
        }

        public static List<Customer> FindCustomersByCity(
            List<Customer> customers,
            string city)
        {
            return customers.FindAll(c => c.City == city);
        }

        static List<Customer> CreateCustomers()
        {
           return new List<Customer>
           {
        new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
        new Customer(2) { Name = "Laurence Lebihan",  City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",   City = "London"    },
        new Customer(4) { Name = "Ann Devon",         City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",     City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",       City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",    City = "London"    },
        new Customer(8) { Name = "Liz Nixon",         City = "Portland"  }
           };
        }
    }
}
```

Understanding Queries and Query Expressions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Expressions;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
```

LINQ Language Integrated Query

```
class Program
    {
        static void Main(string[] args)
        {
            Func<int, int> addOne = n => n + 1;
            Console.WriteLine("Result: {0}", addOne(5));

            Expression<Func<int, int>> addOneExpression = n => n + 1;

            var addOneFunc = addOneExpression.Compile();
            Console.WriteLine("Result: {0}", addOneFunc(5));
        }

        public static List<Customer> FindCustomersByCity(
            List<Customer> customers,
            string city)
        {
            return customers.FindAll(c => c.City == city);
        }

        static List<Customer> CreateCustomers()
        {
           return new List<Customer>
           {
        new Customer(1) { Name = "Maria Anders", City = "Berlin"},
        new Customer(2) { Name = "Laurence Lebihan", City = "Marseille" },
        new Customer(3) { Name = "Elizabeth Brown",  City = "London"    },
        new Customer(4) { Name = "Ann Devon",        City = "London"    },
        new Customer(5) { Name = "Paolo Accorti",    City = "Torino"    },
        new Customer(6) { Name = "Fran Wilson",      City = "Portland"  },
        new Customer(7) { Name = "Simon Crowther",   City = "London"    },
        new Customer(8) { Name = "Liz Nixon",        City = "Portland"  }
           };
        }
    }
}
```

Anonymous Types and Advanced Query Creation

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Expressions;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Store
    {
        public string Name { get; set; }
        public string City { get; set; }

        public override string ToString()
        {
            return Name + "\t" + City;
        }
    }
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
```

```
class Program
    {
        static void Query()
        {
            var stores = CreateStores();
            var numLondon = stores.Count(s => s.City == "London");
            Console.WriteLine("There are {0} stores in London. ", numLondon);
        }

        static void Main(string[] args)
        {
            Query();
        }

        public static List<Customer> FindCustomersByCity(
            List<Customer> customers,
            string city)
        {
            return customers.FindAll(c => c.City == city);
        }

        static List<Store> CreateStores()
        {
           return new List<Store>
           {
              new Store { Name = "Jim's Hardware",    City = "Berlin"     },
              new Store { Name = "John's Books",      City = "London"     },
              new Store { Name = "Lisa's Flowers",    City = "Torino"     },
              new Store { Name = "Dana's Hardware",   City = "London"     },
              new Store { Name = "Tim's Pets",        City = "Portland"   },
              new Store { Name = "Scott's Books",     City = "London"     },
              new Store { Name = "Paula's Cafe",      City = "Marseille"  }
           };
        }
        static List<Customer> CreateCustomers()
        {
           return new List<Customer>
           {
              new Customer(1) { Name = "Maria Anders", City = "Berlin"},
              new Customer(2) { Name = "Laurence Lebihan",City = "Marseille"},
              new Customer(3) { Name = "Elizabeth Brown", City = "London"},
              new Customer(4) { Name = "Ann Devon",  City = "London"},
              new Customer(5) { Name = "Paolo Accorti", City = "Torino"},
              new Customer(6) { Name = "Fran Wilson", City = "Portland"},
              new Customer(7) { Name = "Simon Crowther", City = "London"},
              new Customer(8) { Name = "Liz Nixon",City = "Portland"  }
           };
        }
    }
}
```

Service Oriented Architecture (SOA)

Understanding SOA

An architectural concept or style:
uses a set of "services" to achieve the desired functionality.
SOA implements
a set of loosely coupled services that collectively achieve the desired results.
A service is an autonomous (business) system
that accepts one or more requests and returns one or more responses
via a set of published and well defined interfaces.

Design Considerations in SOA

What services do you need?
What services are available for you to consume?
What services will operate together?
What substitute services are available?
What dependencies exist between services and other versions of services?

Service Orientation

A service is based on four fundamental tenets.

Tenet 1: Boundaries Are Explicit

Know your boundaries:
A well-defined and published public interface is the main entry point into the service.
Services should be easy to consume:
It should be easy for other developers to consume the service. (without breaking existing consumers)
Keep the service surface area small:
Provide fewer public interfaces that accept & responds with a well-defined message.
Don't expose implementation details:
These should be kept internal (loosely coupled)

Tenet 2: Services Are Autonomous

Service versioning and deployment are independent of the system in which they are deployed.
Contracts, once published, should not be changed.

Tenet 3: Services Share the Schema and Contract, Not the Class

Service contracts constitute
data, WSDL, and the policy
Service contract do not change and remain stable.
Contracts should be as explicit as possible

Tenet 4: Service Compatibility Is Based on Policy

Policy separate:
"what is communicated" &
"how/whom a message is communicated"
Policies should be used
to be able to express all the requirements of service interaction via description mechanism (WSDL) alone.

The Connectivity Imperative

Interoperability & Integration
Secure, Reliable, Transacted Messaging
Decoupled, Dynamic Applications

One Question to all Software Developers

"What would it take for my software to be an asset?"
-Build it to communicate
"I should not be afraid of changing it."
-Provide a thoroughly explicit interface
"I should be able to keep on using it."
-Seamlessly connect into any topology
"I want to be able to see that its working."
-Have rich administration facilities

WCF Services

Windows Communication Foundation

Productivity	Unifies today's distributed technologies Attribute-based development Visual Studio integration
Interoperability	Broad support for WS-* specifications Compatible with existing MS distributed application technologies
Service-Oriented Development	Enables development of loosely-coupled services Config-based communication

Unified Programming Model

Unified Model Benefits

- Programming model
  - Learning curve
  - Consistency
  - Write less code when using multiple technologies
- Flexibility
  - Environments
    - Productivity in development environment
    - Simplify Automated Integration Testing
    - Deployment options in production
      - Design for distribution, run local

Unified Programming Model

- WCF extends the .NET Framework
- Services are built in Visual Studio using any .NET programming language

WS-* Protocol Support

From Objects to Services

Four Tenets of Service Orientation

WCF Services

WCF Services

WCF Architecture

WCF Services

Service Endpoints

Address
Where the endpoint can be found?

Binding
How the client communicates with the endpoint?

Contract
What functionality is exposed to the client?

Behaviour
Implementation details

Simple WCF Service

```
[ServiceContract]
public interface IMyInterface
{
[OperationContract]
void MyMethod();
}
```

Contract Definition

```
[ServiceBehavior(InstanceContextMode=Single]
public class MyService: IMyInterface
{

[OperationBehavior(Impersonation =
        ImpersorationOption.Required)]
public void MyMethod() {/* do something */}
    }
```

Service Implementation

```
<service name="MyService">
<endpoint
address="net.tcp://localhost:1234/MySvc"
binding="netTcpBinding"
contract="IMyInterface" />
```

Endpoint Definition
Address+Binding+
Contract

WCF **Address**

Service Location

net.pipe://myserver/stockquoteservice
net.tcp://myserver:8080/stockquoteservice
http://myserver/stockquoteservice/
net.msmq://myserver/private/stockquoteresponse

WCF Binding

Binding represent wire-level agreements between a client and a server
Binding is a preconfigured channel stack
Binding specifies
- Transport
- Encoding
- Security

Service Programmer: provide Host

```
                //Self Hosting
ServiceHost serviceHost = new ServiceHost(typeof(StockService),
                          new Uri("http://localhost:8000/Stock"));
serviceHost.AddServiceEndpoint(
                                typeof(IStockService),
                                new BasicHttpBinding(),
                                "");
ServiceMetadataBehavior behavior = new ServiceMetadataBehavior();
behavior.HttpGetEnabled = true;
serviceHost.Description.Behaviors.Add(behavior);
serviceHost.AddServiceEndpoint(
                typeof(IMetadataExchange),
                MetadataExchangeBindings.CreateMexHttpBinding(),
                "mex");
serviceHost.Open();
Console.WriteLine("The services is ready. Press <ENTER> to terminate.\n\n");
Console.ReadLine();
// Close the ServiceHostBase to shutdown the service.
serviceHost.Close();
```

Service Programmer: provide Host

```
//Self Hosting
 ServiceHost serviceHost = new ServiceHost(typeof(StockService));
 serviceHost.Open();
 // The service can now be accessed.
 Console.WriteLine("The services is ready. Press <ENTER> to terminate.\n\n");
 Console.ReadLine();
 // Close the ServiceHostBase to shutdown the service.
 serviceHost.Close();
;
```

Service Programmer: Provide Host

```
//IIS Hosting
//Within IIS 5.1 or 6 (HTTP only) or 7 (any transport)
//DealAnalyzer.svc
<%@Service Class="WoodgroveBank.DealAnalyzer" %>
<%@Assembly Name="DealAnalyzerAssembly" %>
```

Service Administrator: Configure

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="Stock.StockService"
              behaviorConfiguration="myServiceBehavior">
            <endpoint address=" http://localhost:8000/Stock "
              binding="basicHttpBinding"
              contract="EssentialWCF.IStockService" />
            <endpoint address="mex"
              binding="mexHttpBinding"
              contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="myServiceBehavior">
              <serviceMetadata httpGetEnabled="True"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Client Programmer: Invoke service

```
[ServiceContract]
        public interface IStockService
        {
```

```
        [OperationContract]
        double GetPrice(string ticker);
    }
ChannelFactory<IStockService> myChannelFactory =
            new ChannelFactory<IStockService>(
            new BasicHttpBinding(),
            new EndpointAddress("http://localhost:8000/Stock"));
IStockService wcfClient = myChannelFactory.CreateChannel();
double p = wcfClient.GetPrice("msft");

Console.WriteLine("Price:{0}",p);
```

Client Programmer: Invoke service

Download Metadata
Generate Typed Proxy
Generate Address
Generate Binding

SvcUtil.exe "http://localhost:8080/Deals/AnalysisService"

```
        /out:DealAnalysisProxy.cs /config: app.config
Invoke Typed Proxy Methods
  StockServiceClient proxy = new StockServiceClient();
        double p = proxy.GetPrice("msft");
        Console.WriteLine("Price:{0}", p);
        proxy.Close();
```

What is WCF Contract?

WCF Contracts

Service	Describe the operation a service can perform.
Data	Describe a data structure.
Message	Defines the structure of the message on the wire

A contract specifies:

Operations can be called by a client.
The type of input parameters or data required to call the operation.
What type of processing or response message the client can expect.
Structure of the message
Behavior of the service

Types of Contract

ServiceContract:

Defines Operations, Behaviors
What does your service do?

OperationContract:
Applied on methods which are available for client

DataContract:
Used for Serializing Custom Defined types from service to client.
Defines Schema and Versioning Strategies
What object data needs to flow back and forth?

MessageContract:
Used for embedding custom types in message.
Allows defining application-specific headers and body content
Allows control over the SOAP structure of messages

FaultContract:
Used for exposing exceptions from service to client.

Service Contract

```
using System.ServiceModel;
[ServiceContract]
public interface ICalculate
    {
[OperationContract]
double Add( double a, double b);
[OperationContract]
double Subtract( double a, double b);
    }
```

Service Contract: Duplex

```
[ServiceContract(Session=true,
CallbackContract=typeof(ICalculatorResults)]
public interface ICalculatorProblems
{
[OperationContract(IsOneWay=true)]
void SolveProblem (ComplexProblem p);
}
public interface ICalculatorResults
{
[OperationContract(IsOneWay=true)]
void Results(ComplexProblem p);
}
```

[OperationContract]

Defines operations provided by the service

Properties
IsOneWay
IsInitiating
IsTerminating
Action

Client Service Interaction

One Way:
Datagram-style delivery
Request-Reply
Immediate Reply on same logical thread
Duplex
Reply "later" and on backchannel (callback-style)

Operation Contract: OneWay

```
[ServiceContract]
public interface IOneWayCalculator
{
[OperationContract(IsOneWay=true)]
void StoreProblem (ComplexProblem p);
}
```

Data Contract

[DataContract]

```
public class ComplexNumber
        {
    [DataMember]
    public double Real = 0.0D;
    [DataMember]
    public double Imaginary = 0.0D;

    public ComplexNumber(double r, double i)
        {
        this.Real = r;
        this.Imaginary = i;
        }
        }
```

Serialization

WCF supports two serializers out-of-the-box
DataContractSerializer (DCS)
XmlSerializer (XS)
Custom serializers can be written and hook it in Services

DataContractSerializer

DataContractSerilizer supports
CLR Primitive Types: Int32, Decimal, Boolean
Other Primitive Types: Byte Array, DateTime, etc
Enums
[DataContract], [CollectionDataContract]
IXMLSerializable
Arrays and Collections
[Serializable], Iserializable
Generic and Nullable Types
Suppcrts Classes, interfaces (as Object) and structures

Not supported
XMLElementAttribute and XMLAttributeAttribute
sUse XmlSerializer instead

DataContract Serializable Objects

Data Contract
- Decorated public and private fields and properties
- In System.Runtime.Serialization namespace
- [DataContract] attribute required

Serializable
- All public and private fields, except [NonSerialized]
- Use [OnDeserializing] instead of the constructor
- [Serializable] attribute required
- ISerializable optional

XML Serializable
- Must implement IXmlSerializable.

Collections

Collections are interchangeable
All list collections of the same type are considered to have the same data contract

```
[DataContract(Name = "PurchaseOrder")]
public class PurchaseOrder1
{
    [DataMember]
    public Collection<Item> Items;
}
```

```
[DataContract(Name = "PurchaseOrder")]
public class PurchaseOrder2
{
    [DataMember]
    public List<Item> Items;
}
```

Same applies to dictionaries of the same type
Concrete classes and interfaces

[CollectionDataContract]
More control over collection serialization

[MessageContract]

```
[MessageContract]
public class ComplexProblem
      {
      [MessageHeader]
public string operation;
      [MessageBody]
public ComplexNumber n1;
      [MessageBody]
public ComplexNumber n2;
      [MessageBody]
public ComplexNumber solution;
// Constructors…
}
```

[FaultContract]

```
[ServiceContract(Session=true)]
      public interface ICalculator
            {
[OperationContract]
[FaultContract(typeof(DivideByZero))]
ComplexProblem SolveProblem (ComplexProblem p);
            }
;

try {

 return n1 / n2;
      }
catch (DivideByZeroException e) {
DivideByZero f = new DivideByZero ("Calculator");
throw new FaultException<DivideByZero>(f);
}
```

WCF Binding

Bindings & Channels

A Binding is composed of a set of Binding Elements.
Each Binding Element belongs to a Binding Layer
Binding Layer defines
Transport
Encoding
Protocol
A Channel is a concrete implementation of a Binding Element.

Defining Endpoints

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.serviceModel>
    <services>
      <service serviceType="CalculatorService">
        <endpoint address="Calculator"
                  binding="basicHttpBinding"
                  contract="ICalculator" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Bindings

Binding	Security[Default]	Session[Default]	Transaction	Duplex
BasicHttpBinding	None,Transport,Message	None	n/a	
WSHttpBinding	Transport,Message,Mixed	None,Transport,Reliable Session	n/a	
WSDualHttpBinding	Message	Reliable Session	Yes	
WSFederationHttpBinding	Message	None,Reliable Session	No	
NetTcpBinding	Transport,Message	Reliable Session,Transport	Yes	
NetNamedPipeBinding	Transport	None,Transport	Yes	
NetMsmgBinding	Message,Transport,Both	None	No	
NetPeerTcpBinding	Transport	None	Yes	
MsmgIntegrationBinding	Transport	None	n/a	

Web Service Bindings

BasicHttpBinding
for Simple Object Access Protocol(SOAP) version1.1 compatibility
WSHttpBinding
for SOAP version 1.2 with WS*
WSDualHttpBinding
for callbacks over HTTP with WS* support (not interoperable)
WSFederationHttpBinding
for federated security and single sign-on (SSO) scenarios

Cross-Process / Machine Bindings

NetNamedPipeBinding
for in-process or same-machine calls
NetTcpBinding
for same-machine or cross-machine calls
NetPeerTcpBinding
for same-machine or cross-machine peer-to-peer messaging

Messaging Bindings

NetMsmqBinding
for reliable, transacted, and persistent messaging over MSMQ
MsmqIntegrationBinding
for MSMQ interoperability with earlier technologies

NetTcpBinding

Default binding is faster than http-based
Binary-encoded SOAP directly over TCP

NetMsmqBinding

Requires one way operations
Messages sent to the message queue rather than the endpoint
Multiple WCF messages can be mapped to one MSMQ message if SessionMode= Required.
Binary-encoded SOAP over MSMQ

NetNamedPipeBinding

Binary-encoded SOAP over named pipes
communication between processes on the same Windows machine.

Custom Bindings

```
            <bindings>
            <customBinding>
        <binding configurationName="Binding1">
            <reliableSession bufferedMessagesQuota="32"
                inactivityTimeout="00:10:00"
                maxRetryCount="8"
                ordered="true" />
            <httpsTransport manualAddressing="false"
                maxMessageSize="65536"
                hostnameComparisonMode="StrongWildcard"/>
            <textMessageEncoding maxReadPoolSize="64"
                maxWritePoolSize="16"
                messageVersion="Default"
                encoding="utf-8" />
        </binding>
    </customBinding>
</bindings>
```

Prasanna Wadekar

Custom Binding

```
System.ServiceModel.Channels.Binding customBinding= new CustomBinding (
        new SecurityBindingElement(),
        new MtomMessageEncodingBindingElement(),
        new TcpTransportBindingElement());
```

Custom
- Use CustomBinding
- Fill with binding elements

Inherit
- Derive from system-provided bindings

User-defined
- Derive from Binding
- Add your own or system-provided elements

Bindings & Binding Elements

WCF Services

WCF Hosting

Service Host is an Operating System Process
- Responsible for lifetime and Service Context of WCF Service.

Service host knows very little about the WCF service that runs in its memory space
- Any operating system process can be a service host.
  - Console, Windows Forms, WPF Application
  - Windows NT Service
  - IIS
  - WAS (Windows Activation Service)

Self-Hosting

Console, Windows Forms, WPF Application
A ServiceHost instance must be initialized to expose endpoints for a service
Core methods:
- Open() – opens channel listeners
- Close() – Closes channel listeners

Service Clients

Through code or configuration
Service knowledge
- Extract metadata
  - SvcUtil.exe
  - Custom tool
  - Generates code and config

Client Typed Proxy

ClientBase<TChannel>

```
Internal class MyCustomerServiceClient: Client<IMyCustomerService>,
IMyCustomerService
    {
public MyCustomerServiceClient(string endpointConfigurationName)
      {
      }

public MyCustomerClient(Binding binding,EndpointAddress endpointAddress)
:base(binding,endpointAddress)
{
}
```

ChannelFactory<TChannel>

```
internal class MyCustomerServiceClient:ClientBase<IMyCustomerService>,IMCustomerService
            {
            }
ChannelFactory<ICustomerService>.CreateChannel(new NetTcpBinding(), new
EndpointAddress(NETTCP_URI + SERVICE_NAME));
```

Hosting Multiple Services

```
                    public static void Main()

                    {
ServiceHost serviceHost1 = new ServiceHost(typeof(GoodStockService));
            serviceHost1.Open();
ServiceHost serviceHost2 = new ServiceHost(typeof(GreatStockService));
            serviceHost2.Open();
            .......
            .......
            serviceHost1.Close();
            serviceHost2.Close();
}
```

Windows Applications

- Windows Forms or WPF
- Exposing services from client machines
- Must be aware or synchronization context
  - UI thread or not
- If ServiceHost is opened on a non-UI thread, services operate on a new thread
- If on UI thread, services join it unless UseSynchronizationContext set to false

Windows Services

- Host environment initialized
  - when machine starts, can be configured to restart on failure
- Open and close ServiceHost instances on start and stop of the Windows Service

IIS Hosting

- IIS provides Hosting Infrastructure
  - Process activation
  - Recycling
  - Identity management
- Steps for IIS Hosting
  - create an IIS Virtual Application,
  - create an SVC file to define the service implementation
  - Include <System.ServiceModel> section in web.config

WAS Hosting

- Scenarios in which services to be developed for
  - Offline client server interaction (disconnected environment)
  - Chat services (quickly sending numerous small messages as a part of a larger conversation)
- IIS Hosting can not provide above functionality
- Windows Activation Service (WAS)
  - WAS is the hosting infrastructure built into Vista and Windows Server 2008
  - WAS supports multiple protocols through a listener adapter architecture

WAS can communicate over HTTP, TCP, MSMQ, and namedpipes using a consistent mechanism, thereby improving system reliability

Hosting a service in WAS is similar to hosting in IIS.

Additional Steps:

Install (or confirm the installation of) the WCF activation components.

Configure WAS to support a non-HTTP protocol.

Install (or confirm the installation of) the WCF activation components.

To install the WCF non-HTTP activation components

Click the Start button, and then click Control Panel.

Click Programs, and then click Programs and Features.

On the Tasks menu, click Turn Windows features on or off.

Find the .NET Framework 3.5 node, select and then expand it.

Select the WCF Non-Http Activation Components box and save the setting.

Configure WAS to support a non-HTTP protocol.

To support net.tcp activation, the default Web site must first be bound to a net.tcp port

```
%windir%\system32\inetsrv\appcmd.exe set site "Default Web Site" -
+bindings.[protocol='net.tcp',bindingInformation='808:*']
```

enable net.tcp for the application

```
%windir%\system32\inetsrv\appcmd.exe set app "Default Web Site/<WCF
Application>" /enabledProtocols:http,net.tcp
```

Prasanna wadekar 9970248407

* **Languages**
  - **compiled**
    - compiler is required
    - compiles the whole source code into an executable
    - errors are detected at the time of compilation
    - compilation cannot proceed with error
    - faster
    - eg: c, c++
  - **interpreted**
    - interpreter is required
    - interprets the code line by line
    - errors are detected at run time (executing the code)
    - execute can not proceed with error
    - slower
    - e.g.: HTML, JavaScript, (all scripting languages)

* **Applications**
  - web applications (web site)
    - browser is compulsory
    - mainly HTML
    - browser can understand HTML
    - platform indenpendant
    - slower
    - eg: HTML
  - **Native applications**
    - OS + CPU
    - mainly ASM language
    - CPU can understand only ASM
    - platform specific
    - faster
    - eg. C,C++

PRASANNA

* **Web Application**
  - HTML: design UI
  - CSS: to add decoration in HTML code
  - JavaScript: to add programming logic in HTML

* **HTML**
  - ML: tags + data (content within tags)
    - **tag:**
      - word enclosed in < and >
      - types
        - starting tag:
        - closing tag:
        - empty tag:
</br> =

- root tag:
  - also known as document type/tag/element
  - tag which starts a document and which ends the document

- **Parts**
  - head
    - title: shows the title on tab/window
    - meta: to add extra information (
    - style: to add CSS styles
    - link: to link external documents
    - script: to add JavaScript code
    - base: to decide the base url of the website
      - to configure the anchor tags
  - **body**
  - Text Elements
    - inline
      - these elements do not create new line
      - eg: span, superscript etc
    - **block**
      - these elements create a new line at the end
      - eg: para, div, h1
  - **Link elements**
    - anchor
  - Table
    - head
    - body
    - footer
  - **List**
    - ul
    - ol
    - definition list (dl)
  - **Form**
    - input
    - select
    - textarea

* **Default Attributes**
- style: to add the CSS properties
- id: to identify the tag uniquely
- class: to identify similar type of element having same requirements

* **CSS**

- Cascading Style Sheet
- is introduced in HTML 4.0 to add decoration
- CSS3
- **Types**

- inline style
  - to add decoration in a tag
  - **Disadvantage:**
    - repeat the same info for every tag
- **Internal Style**
  - to target multiple elements at once
  - internal to the current document
  - disadvantage:
    - repeat the same info for every page
- **External Style**
  - to target multiple pages at once
  - link tag is used to link external css file with the html document
  - eg: <link href=" <file name>" type=" text/css" rel=" stylesheet" >
- browser default style
  - this style is by default applied on every html page
  - this style is browser dependant

- **Order**
  - browser default style: 1
  - external style: 2
  - internal style: 3
  - inline style: 4

*** CSS selector**
- which can target multiple elements/tags at once
- types of selector
  - element selector
    - selects similar type of elements
    - e.g.: div{color:red;} -> all div elements will be in red color
  - group selector (,)
    - can select multiple type of elements
    - e.g.: div, p {"} -> all div and para will be targeted
  - id selector (#)
    - to target only a specific tag (having specific id)
    - eg: div#specialdiv {..}
  - class selector (.)
    - to target elements having same class name
    - eg: div.greendiv {..}
  - pseudo selector: (:)
    - which gets executed in a specific condition
    - eg:
      - :hover -> when mouse pointer is on top of this element
      - :nth-child -> for n-child

CSS Box Model →

CSS Rule →

***JavaScript**

- is a interpreted language
- browser acts as interpreter
- is a object based language
- usage:
  - to add programming logic in HTML
  - these days it is also used by third party software like Unity3D to add programming logic in mobile games
  - these days it is also used on server to communicate with Web server to create dynamic web pages

*** Variables**

- data type can NOT be assigned in JavaScript
- JS decides the data type by looking at the variable"™s CURRENT value
- variables can be redeclared

- eg:

```
name = "sunbeam"□;  // string
salary = 6.7;      // number
count = 10;        // number
canVote = true;    // boolean
```

- **Scopes**
  - Global scope
    - create a variable without using var keyword inside a function
    - or declare a variable outside any function
    - e.g.: name = "sunbeam"□;

  - Local scope
    - declare a variable with var keyword inside a function
    - e.g.: var myname = "sunbeam"□;

  - var keyword is used to declare a variable
    - if var is used inside a function: it creates local var
    -

```
function function1() {

var myname = "test"□; // local
}
```

    - if var is used outside a function: it create global var
    -

```
var myname = "test"□; // global

function function1() {
}
```

- **function**
  - function is a keyword used to declare a function
  - syntax:

```
function <function name> (<number of parameters>) {
    // function body
}
```

  - a function can be called by passing variable number of parameters
    - if less number of params are passed: then remaining will be assigned undefined value
    - if more number of params are passed: then extra params will be ignored
    -

```
function add(num1, num2) {
  console.log("num1 = "□ + num1);
  console.log("num2 = "□ + num2);
```

prasanna wadekar


```
}

add();        // num1 = undefined, num2 = undefined
add(10);      // num1 = 10, num2 = undefined
add(10, 30);  // num1 = 10, num2 = 30
add(1,2,3,4); // num1 = 1, num2 = 2: rest of the params will be ignored
```

- **JS pre-defined words**
  - undefined:
    - a var is declared but not defined
    - the value is not set
  - NaN:
    - not a number
    - mathematical operations are performed on a non-number value
    - eg: undefined + undefined = NaN
      undefined + 10 = NaN
      "test "□ * "test "□ = NaNs

- **Browser**
  - special application which can understand HTML
  - it is used to render/display html pages
  - **components**
    - UI component
    - data storage
      - store data
      - cookies
      - HTML5
        - session storage
        - local storage
    - networking component
    - javascript engine
    - rendering engine
      - Apple Safari: webkit
      - Google Chrome: webkit -> Blink
      - Mozilla Firefox: Gecko
      - Opera: gastro
      - IE: -

* **URL**

- Uniform Resource Locator
- to identify any resource from internet uniquely
- contains components
  - scheme: (http)
    - generally known as protocol

- decided how to fetch the resource from internet
- eg: http, https, file, ftp, tftp etc
- domain information: (www.mywebsite.com)
  - www: sub-domain
  - mywebsite: domain name (gets resolved into a unique IP address by DNS)
  - com: top level domain
- port number: (80)
  - identifies the service running on the server
  - if port number is not provided then the default port number for specified scheme is used
  - eg: http:80, https:443, ftp:21
- file / path: (myfolder/myfile.html)
  - path to the resource
- query string: (firstname=test&seconname=test)
  - is the way to provide input to a page
  - it always present in key=value format
  - e.g.: firstname=test and secondname=test
    where: firstname, secondname are keys
    test, test are values
  - multiple key-value pairs are separated by &
- hash (#) component: (#top)
  - used to link a part of same document

eg: http://www.mywebsite.com:80/myfolder/myfile.html?firstname=test&seconname=test
eg: http://www.mywebsite.com:80/myfolder/myfile.html#top

*** Object**

- collection of different data members (properties) and functions associated with them
- **2 ways to create an object**
  - by using new Object
    - Object is a kind of template given by JavaScript
    - eg:

```
var person = new Object();    <- Object
person.name = "test"□;        <- property
person.address = "Karad"□;    <- property
console.log("name : " + person.name);

var person1 = new Object();
person1["name"] = "test1";
person1["address"] = "Pune";
console.log("name : " + person["name");
```

  - json
    - JavaScript Object Notation
    - the simplest way to create an object

- these days, json is used to transfer data from one entity to another
  - used to send data from server to client (mobile Facebook application)
  - because
    - json is light weight text file
    - easy to parse (extracting information) json response
- eg:

```
var person = {
    "name":"test" ,
    "address":"karad"
};
console.log("name : " + person.name);
```

- **DOM**
  - Document Object Model

- hierarchy of tags
- W3C has 3 types
  - Core DOM
    - language independent
  - HTML DOM
  - XML DOM

- **JavaScript built in objects**
  - these objects are created by browser at the time of loading the page
  - document
    - representation of HTML DOM
      - using DOM object (document), html tags can be accessed/modified at run time
    - it contains JS object of every html tag inside the html file
    - using document any html tag can-be accessed by using:
      - getElementById(): returns a tag which has the same id as that of the input
      - getElementsByTagName(): by matching the tag names
      - getElementsByName(): by matching the name attribute

- getElementsByClassName(): by matching the class name
- every tag contains property named innerHTML which can be used to set / get the contents of that tag except input (value)
- sessionStorage:
- localStorage:
- navigator:

***** HTML5 functionality *****

*** Web Storage**
- used to store data temporarily or permanently
- key-value pairs can be stored in side web storage
- only ASCII character (string) can be stored in web storage
- implemented and maintained by browser internally
- stored in different location for different browser
- **Types**
  - session storage
    - stores the data temporarily
    - till the session is alive
    - session:
      - the duration of browser
      - it gets created at the time of opening browser and gets closed at the time of closing browser
  - local storage
    - data gets stored permanently
    - data upto 5MB can be stored in local storage

*** Geolocation**
- to get the current location in terms of Latitude and Longitude
- ways (providers) to retrieve the current position
  - GPS hardware
  - IP address
  - cell triangulation (A-GPS)

*** Multimedia**
- video

*** Canvas**

*** Pop up box**
- alert
  - used to display readonly message
  - e.g. alert("hello..");
- prompt
  - used to get a single line input from user
  - e.g. var name = prompt("enter name..");
- confirm

- used to get confirmation from user
- e.g. var answer = confirm("are you hungry?"); // true or false

*** Date object**

*** timer functions**
- used to perform functionality after an interval
- setInterval()
  - used to call a function every after some interval
  - setInterval(<function name>, <interval in mili-seconds>);
- clearInterval()
  - stops calling a function which is started by setInterval()
- setTimeout()
  - used to call a function only once after few mili-seconds
  - e.g. setTimeout(<function name>, <interval in mili-seconds>);
- clearTimeout()
  - stops calling the function which is started by setTimeout()

*** jQuery**
- javascript library of commonly used functions
- benefits
  - free and open source
  - has a very good community

*** XML**
- is a Markup Language
  - made up of elements and data
- eXtended Markup Language
- standardized language by w3c
- imp
  - the elements are user-defined: xml writer must define the xml elements
  - simple text file format
  - xml is interpreted
  - an interpreter is require: by default browser is used to interpret the xml
- usage
  - structuring the data
  - to transfer data from one entity to another in commonly understandable format
  - to store very small amount data
- xml vs html
  - html is used for designing user interface for browser
    xml is NOT used to design UI
  - html has pre-defined tags
    xml has user-defined elements
  - html is case in-sensitive : <table></Table>
    xml is case sensitive

* **XML element**
  - word enclosed in < and >
  - types
    - starting element: <name>
    - closing element: </name>
    - empty element
      - normal syntax: <name></name>
      - shorthand: <name />
    - root element
      - which starts and closes the document
      - also known as document type or document element
  - rules to create an element
    - every file must have one and only one root element
    - Special characters like space can not be used in element
      - <First Name> <- incorrect element
      - <FirstName>, <First_Name>
    - only _ can be used in element
      - <First_Name>
    - elements cannot start with number
      - <1Name> <-- incorrect element-->
      - <Name>
    - every starting element must be closed with closing element
      - <Name>test <- incorrect -->
      - <Name>test</Name>
    - element name is case sensitive
      - <Name> and <name> are two different element

* **Attribute**
  - it adds more information in xml element
  - it is presented in name=value pair
    - e.g. phoneType="work"
  - rules
    - can not be repeated
      - e.g. <Phone phoneType="work" phoneType="special"></Phone>
        <Phone phoneType="work"></Phone>
    - in name=value format
    - can NOT have multiple values at the same time
      - e.g. <Phone phoneType="work","special"></Phone>
    - can NOT have a child attribute
  - attribute-only element
    - having only attributes
      - e.g. <Person name="test" address="Karad" />
  - mixed type element
    - has both: child element and attribute
    - <Person name="test" address="Karad">
      <Phone phoneType="special">4544</Phone>
      <Phone phoneType="work">4544</Phone>

<Phone phoneType="personal">45464</Phone>
</Person>

*** Parts**
- XML Header
  - <?xml version="1.0" encoding="UTF-8" ?>
  - it is optional
  - where:
    - version: specification version used to create this xml
    - encoding: type of characters using in this xml
      - UTF: Unicode Transformation Format
        - UTF-8: 8 bit or 1 byte
        - UTF-16: 16 bit or 2 bytes (wide character)
- XML body
  - collection of user-defined elements

*** CDATA**
- the characters included in the CDATA section are not parsed
- special meaning of these characters will not be used while presenting xml
- <![CDATA[> <data> <]]>

* Well formed xml document
- the xml is following all the syntactical rules specified by W3C

*** Valid XML**
- a well formed xml which follows all the logical rules as well
- logical rules can be defined by using DTD or XML Schema
- every Valid xml must be well formed xml but every well formed xml may NOT be valid xml

*** DTD**
- Document Type Definition
- used to define the logical rules
- types
  - a root element (!DOCTYPE)
    - used to defined a root element
    - <!DOCTYPE <element name> [<logical rules>]>
  - elements
    - simple element (!ELEMENT)
      - <!ELEMENT <element name> (#PCDATA)>
      - where
        - #PCDATA: all parcelable characters
    - element having child element(s) where order is imp
      - <!ELEMENT <element name> (<child element1>, <child element2>...)>
      - element is made up of sequence of child elements
      - child elements must be present in the pre-defined order
    - element having child element(s) where order is not imp
      - <!ELEMENT <element name> ANY>

Prasanna Wadekar

- where
  - ANY: any element in any order
- empty element
  - element having no data
  - <!ELEMENT <element name> EMPTY>

- **Wild characters**
  - *: element can appear zero or more times
  - +: element can appear one or more times
  - ?: element can either appear only once or absent

- **attributes**
  - !ATTLIST is used to define an attribute
  - <!ATTLIST <element name>
    <attribute name> CDATA <type> >
  - e.g.

```
<!ATTLIST Phone
        phoneType CDATA #IMPLIED>
```

where
- type = #IMPLIED -> optional -> ignore
- type = "value" -> default value -> default value
- type = #REQUIRED -> compulsory

*** DTD**

- Types
  - internal DTD
    - the DTD definition written in the same xml file
    - the file becomes cluttered with xml content and DTD content
  - external DTD
    - a new file with .dtd extension must be created with all the definitions
    - external DTD can NOT start with DOCTYPE
    - to load external DTD SYSTEM keyword is used
    - <!DOCTYPE <element name> SYSTEM "<dtd file name>">
  - mixed DTD
    - combination of both internal and external DTD
    - <!DOCTYPE <element name> SYSTEM "<dtd file name>" [
      <internal DTD rules>
      ]>

- **Limitations**
  - DTD has its own syntax
  - DTD can not differentiate between character and number
  - values can not be restricted using DTD
  - DTD can not add any restriction on type of data and number of times it gets repeated

- DTD can not understand the xml namespace

*** xml namespace**
- group which separates multiple elements having same logical structure
- xmlns keyword is used
  - syntax:
    <table xmlns:<prefix>="http://<namespace name>"></table>
- to put an element inside a namespace
  - <prefix:<element name>>
  - e.g. <p1:table>..</p1:table>
- rules
  - every namespace can be used inside the same element or child elements where it is defined
  - by default all the child elements are the part of same namespace of their parent element
  -

*** XML Schema**
- xml schema is another way to define logical rules
- data types
  - string: accepts any character including number and special symbols
  - integer: positive and negative
  - decimal: float values
  - positiveInteger:
  - negativeInteger:
  - date:
  - time:
  - dateTime:
  - positiveDecimal:
  - negativeDecimal:
- custom types
  - add restrictions

- **element Types**
  - Simple element
    - does NOT contain any child element
    - does NOT contain any attribute
  - Complex element
    - may contain at least a single child element
    - may contain at least a single attribute

- **rules**
  - xml schema must be written outside the xml file (externally)
  - the file extension must be .xsd (xml schema definition/document)

- **Element**
  - use: xs:element
  - syntax:
    <xs:element name="<name of element" type="<element type>" />

- to create custom type

```
<xs:element name="<element name>" >
<xs:simpleType>
<xs:restriction base="<data type>">
<!— add the required condition —>
</xs:restriction>
</xs:simpleType>
</xs:element>
```

- types
  - simple element

```
<xs:element name="<name of element>" type="<element type>" />
```

  - element having child element(s) in a specific order
    - all the child element(s) must be present

```
*   <xs:element name="<element name>" >
<xs:complexType>
<xs:sequence>
<!— child element(s) —>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- element having child element(s) where order is not imp
  - all the child element(s) must be present

```
*   <xs:element name="<element name>" >
<xs:complexType>
<xs:all>
<!— child element(s) —>
</xs:all>
</xs:complexType>
</xs:element>
```

- element having child element(s) where only one child element must be present

```
*   <xs:element name="<element name>" >
<xs:complexType>
<xs:choice>
<!— child element(s) —>
</xs:choice>
</xs:complexType>
</xs:element>
```

- element having repeating child element
  - occurrence indicator
    - minOccurs: minimum number of times the element should me present
      - = 0: element becomes optional
    - maxOccurs: maximum number of times the element should me present
      -> unbounded: infinity

- element having an attribute

*** Path**

- File system path
  - Absolute path
    c:\windows\calc.exe
  - Relative path
    c:\ -> windows\calc.exe
    c:\windows -> calc.exe

*** PHP**

Client – Server Architecture →

- Personal Home Page
- PHP: Hypertext Preprocessor
- **Features**
  - using PHP, dynamic web pages can be designed

- PHP is a scripting language
- PHP is a interpreted language
- Why PHP
  - its very simple to learn and use
  - Open source and free
  - it can be used to design huge web site (having many pages)
  - supports wide range of Database like MySQL, Oracle, SQL Server etc

*** Language Features**

- code must be written inside <?php ?>
  - if page contains only PHP code then ?> is optional
  - if page contains both PHP as well as HTML then ?> is compulsory
- Syntax
  - all the keywords, function names and class names are case in-sensitive
  - variable names are case sensitive
- Variables
  - variable name must start with $
    - $name = "sunbeam";
  - variable can not decide its data type
  - php decides the data type by looking at the current value
- data types
  - string:
    -
  - int: (integer) -> whole numbers
  - float: decimal numbers
  - bool: boolean (true/false)
  - Array:
  - NULL:
  - object: instance of a class
- Array
  - collection of similar or dis-similar data type
  - syntax: $array = array(<value1>, <value2>....);
  - count(): returns the length of array
  - types
    - single dimensional
    - multi-dimensional
    - associative array
- function
  -
- scope
  - local
    - declared inside function
  - global
    - declared outside of any function
    - global variables can NOT be used directly inside function
      - global keyword must be used to use a global variable inside a function

```
$num = 100;
    function function1() {
        global $num;        ///  <- global keyword is required here
        echo "Num = $num";
    }
```

- static
  - that retains it value

* **Superglobals**
- associative array
- $_GET:
  - array which holds all the input parameters sent using GET method
- $_POST:
  - array which holds all the input parameters sent using POST method
- $_REQUEST:
  - holds all the input parameters present in Request
- $_SERVER:
  - holds parameters sent through HTTP Request

Prasanna wadekar 9970248407 18 r

SUNBEAM

Institute of Information Technology

Placement Initiative

Operating System Concepts - Notes

✦ **Operating System Concepts & Linux Programming:**

✦ **Operating System Concepts**

- Introduction to OS
- Introduction to Comp Hardware
- Process Mgmt
- CPU scheduling
- Memory Mgmt
- File & IO Mgmt

✦ **Linux Programming**

- Linux commands
- Shell Scripts
- System Calls

✦ **What is Operating System?**

- OS is intermediary between application programs and computer hardware.
- OS is a resource allocator, which allocates resources (like CPU, RAM, disk, etc) to the running programs as per their requirements.
- OS is a control program, which controls execution of other programs.
- OS CD/DVD = Core OS + Utilities & Applications
- The part of OS which performs basic minimal functionality is called as "Core OS" or "KERNEL".

✦ **OS Functionalities:**

✦ **Process management:**

- Program is a set of instruction given to the machine.
- It is an executable file.
- This file contains exe header, text section, data section, symbol table, etc.
- The exe header contains:
  - magic number : to identify OS in which program can be executed.
  - address of entry point function.
  - info about all the sections in the file.
- When program is executed, the loader (unit of OS) will load it from the disk into the RAM.
- The loader also verifies magic number and file format and if found invalid, program will be terminated.
- For execution of the program OS allocate some memory in RAM which is logically divided into Text, Data, Stack and Heap sections.
- This is called as a "Process". It is formally defined as "Program in Execution".
- To keep info about the process, OS create one more structure, called as "Process Control Block". This struct contains lot of details like:
  - Process Id
  - Exit status
  - Scheduling Info : state, priority, ...
  - Memory Info : page table, ...
  - File Info : Open file desc table, ...
  - IPC info : signal table, ...
  - Execution context (Depends on OS design)
  - Kernel Stack

PRASANNA

- etc.

✦ **CPU Scheduling:**

- CPU scheduler is unit of OS which decides the next process to be executed on the CPU using certain algorithms like FCFS, SJF, RR, etc.

✦ **Memory Management:**

- During compilation, compiler assumes a machine with minimal config (e.g. TC compiler assumes a machine with 8086 CPU and 1 MB RAM); this machine is called as "Virtual Machine".
- Assuming this machine compiler & linker generates addresses for each instruction and data. These addresses are known as "logical addresses" or "virtual addresses".
- During process creation additional virtual addresses are added for runtime section i.e. stack and heap.
- The set of virtual addresses of any process are called as "virtual address space".
- While loading the program into memory, actual locations in the RAM are used, which are called as "physical addresses" or "real addresses".
- The set of physical addresses of any process in the RAM is called as "physical address space".
- CPU always execute a process with its virtual addresses.
- MMU is a hardware unit that converts virtual address of a process (requested by the CPU) into physical address of the process (in the RAM).
- [CPU] -> [MMU] -> [RAM]
- The base and limit of each process is stored in the PCB of that process and will be loaded into the MMU when the process is loaded in CPU (context switch).
- The virtual address requested by the CPU is converted into physical address as shown in diag.

✦ **File & IO Management:**

- File in collection of data or information on secondary storage device.
- File system is used to manage multiple files on the single partition.
- In UNIX/Linux, each device is also treated as file.

✦ **Hardware Abstraction:**

- OS hides the hardware details from the application and end users.
- The changes done in hardware are mostly accepted by the OS without intervention of end users.

✦ **Networking:**

- Data can be transferred from one machine to another machine in some network.
- LAN, MAN, WAN are types of networks.

✦ **Protection & Security:**

- Protecting the system from the internal threats is known as "protection".
- One process cannot access data of another process.
- One user cannot access data of another user.
- Securing the system from external threats like viruses, trojans, worms, etc is known as "security".
- Typical antivirus applications are used to secure the system.

✦ **User Interfacing:**

- User interfacing is optional feature of OS (some embedded OS do not have user interfacing).
- User interfacing is provided by the OS with help of a special program called as "shell".
- This program takes input/commands from the user and get them executed from core OS/kernel. This program is also known as "command interpreter".
- There are two types of user interface/shell:

1. Graphical User Interface:
   - Windows: explorer.exe
2. Command Line Interface:
   - DOS: command.com
   - UNIX/Linux: bsh, bash, csh, ksh, etc.
   - In UNIX/Linux shell runs within terminal program.

✦ **Computer Hardware:**

- CPU executes machine level instruction (in each process).
- Each IO device has its own internal dedicated processing unit called as "IO device controller".
- CPU is connected to the IO devices and memory via "BUS".

✦ **IO Structure:**

- The IO device controller communicates with CPU using special signal called as "Interrupt".
- When interrupt occur. CPU pause current task/process and jumps to IVT.
- In low memory area (1st 1KB of RAM) Interrupt vector table is kept. which contains addresses of interrupt service routines. The ISR contains logic for handling interrupts.
- Then CPU executes appropriate ISR (after getting address from IVT) and handles the interrupt.
- After completion of ISR, CPU resume the pause task/process.

✦ **Types of Interrupts:**

1. Non-maskable Interrupt:

- Interrupts generated from the hardware, which cannot be disabled are called as "NMI".
- Typical error interrupts and other critical interrupts fall in this category.

2. Maskable Interrupt:

- When interrupt arrives, it is possible that CPU is busy in executing some OS critical code. If such code is interrupted, it may cause inconsistency in execution of the OS.
- So during execution of such critical tasks, OS disables interrupts using special assembly language instructions.
- Such interrupts are called as "maskable" interrupts.
- When interrupts are enabled again (after completion of critical task), the arrived interrupts will be visible to CPU and will be handled.
- Such interrupt can generated by hw as well as sw.
- Special software (assembly) instructions can be used to generate software interrupt (TRAP).

  e.g. x86 -> INT instruction
  ARM -> SWI instruction

✦ **Types of IO:**

✦ **Synchronous IO:**

- CPU initiates IO by giving instructions to IO device.
- IO device performs IO and CPU will keep checking whether IO is completed or not. This continuous checking is known as "polling".
- This type of IO is called as "Sync IO" and is easier to implement.
- However in such IO, the CPU is no utilized properly.

✦ **Asynchronous IO:**

- CPU initiate IO by giving instrucion to IO device and continue with execution of some another task.
- When IO device completes IO, it send interrupt to the CPU.
- When interrupt is received CPU can continue to execute the earlier task.
- Such type of IO is called as "Async IO".
- This type of IO increases CPU utilization.
- However in such case, OS need to maintain "Device Status Table".
- This table contains list of all IO devices, their status (busy/idle) and set of processes waiting for each IO device (Waiting queue of IO device).

✦ **Storage Structure:**

- **Memory/Storage:**
  - |- Primary Memory [Directly accessible by CPU] (Volatile)
    - |- CPU registers
    - |- Cache
    - |- RAM (Electronic)
  - |- Secondary Memory [Not accessible to CPU direct] (Non-volatile)
    - |- Hard disk (Magnetic)
    - |- Optical disk (CD/DVD)
- **Data Flow:**
  - CPU Regr <--> Cache <--> RAM <--> Hard disk
- Comparision of storage devices can be done on basis of speed, size and cost.
- **Cache:**
  - Cache is used avoid speed mismatch between CPU and RAM.
  - Data requested by CPU, if not present in cache, it is first copied from RAM to Cache and then from Cache to CPU.
  - If Data requested by CPU, is present in the cache, it is directly taken from there without informing RAM (about data access). This will speed up the data access.
  - If cache is full, the oldest data will be replaced by newer data. Thus cache always contains recent data accessed.

✦ **Booting:**

- If first sector of the storage device/partition contains bootstrap program, then it is called as "bootable device/partition".
- Bootstrap program is different for each OS and can load kernel of that system into the memory.
- Bootloader program displays options for which OS to be started (in case of multiple OS) and depending on user selection starts appropriate bootstrap program.

✦ **Booting Steps:**

- When computer is powered on, programs from the Base ROM (BIOS) are fetched into the RAM automatically.
- The first program from BIOS i.e. POST is executed, which checks whether all devices are functioning properly.
- Then another BIOS program i.e. bootstrap loader is executed, which find the bootable device and starts bootloader program.
- As explained above, bootloader program starts appropriate bootstrap program, which in turn load OS kernel. Thus OS boots.

✦ **Classfication of OS:**

1. Mainframe systems:

✦ **Resident Monitor:**

- Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.

✦ **Batch Systems:**

- The batch/group of similar programs is loaded in the computer, from which OS loads one program in the memory and execute it.
- Thus programs are executed one after another.
- In this case, if any process is performing IO, CPU will wait for that process and hence not utilized efficiently.

✦ **Multi-Programming:**

- In such systems, multiple program can be loaded in the memory.
- The number of program that can be loaded in the memory at the same time, is called as "degree of multi-programming".
- In these systems, if one of the process is performing IO, CPU can continue execution of another program. This will increase CPU utilization.
- Each process will spend some time for CPU computation (CPU burst) and some time for IO (IO burst).
- If CPU burst > IO burst, then process is called as "CPU bound".
- If IO burst > CPU burst, then process is called as "IO bound".
- To efficiently use CPU, a good mix of CPU bound and IO bound processes should be loaded into memory. This task is performed by an unit of OS called as "Job scheduler" OR "Long term scheduler".
- If multiple programs are loaded into the RAM by job scheduler, then one of process need to be executed (dispatched) on the CPU. This selection is done by another unit of OS called as "CPU scheduler" OR "Short term scheduler".

✦ **Multi-tasking OR time-sharing:**

- CPU time is shared amoung multiple processes in the main memory is called as "multi-tasking".
- In such system, a small amount of CPU time is given to each process repeatedly, so that response time for any process < 1 sec.
- With this mechanism, multiple processes (ready for execution) can execute concurrently.
- There are two types of multi-tasking:

1. Process based multitasking: Multiple independent processes are executing concurrently. Sometimes (i.e. on multiple processors) also called as "multi-processing".
2. Thread based multi-tasking OR multi-threading: Multiple parts/functions in a process are executing concurrently.

✦ **Multi-user:**

- Multiple users can execute multiple tasks concurrently on the same systems. e.g. IBM 360, UNIX, Windows Servers, etc.
- Each user can access system via different terminal.
- There are many UNIX commands to track users and terminals.

E.g. tty, who, who am i, whoami, w, etc.

2. Desktop Systems:

- User convinence and Responsiveness
- Windows, Mac OS X, Linux, etc.

3. Multiprocessor Systems:

- The systems in which multiple processors are connected in a close circuit is called as "multiprocessor computer".
- The programs/OS take advantage of multiple processors in the computer are called as "Multi-processing" programs/OS.
- Since multiple tasks can be exeuted on these processors simultaneously, such systems are also called as "parallel systems".

- There are two types of multiprocessor systems:

A. Asymmetric MP:

- OS treats one of the processor as master processor and schedule task for it. The task is in turn divided into smaller tasks and get them done from other processors.

B. Symmetric MP:

- OS considers all processors at equal level and schedule tasks on each processor individually.
- All modern desktop systems use SMP.

4. Distributed Systems:

- Multiple computers connected together in a close network is called as "distributed system".
- It advantages are high availability, high scalability, fault tolerance.
- The requests are redirected to the computer having less load using "load balancing" techniques.
- The set of computers connected together is called as "cluster".

5. Real Time Systems:

- The OS in which accuracy of results depends on accuracy of the computation as well as time duration in which results are produced, is called as "RTOS".
- If results are not produced within certain time (deadline), catastrophic effects may occur.
- These OS ensure that tasks will be completed in a definite time duration.
- Time from the arrival of interrupt till handling of the interrupt is called as "Interrupt Latency". RTOS have very small and fixed interrupt latencies.
- Examples: uC-OS, VxWorks, pSOS, RTLinux, etc.
- Soft real time systems: Not much time critical
- Usually have some secondary storage device.
- Applications: Consumer electronics, ...
- Hard real time systems: Highly time critical
- Usually do not have some secondary storage device.
- Applications: Defence systems, Medical instruments, Space ships, ...

6. Handheld Systems:

- OS installed on handheld devices like mobiles, PDAs, iPODs, etc.
- Challenges:
  - Small screen size
  - Low end processors
  - Less RAM size
  - Battery powered

✦ **OS Design (Implementation):**

1. Simple structure: MS-DOS

```
----------------------------
COMMAND.COM <- command interpreter
----------------------------
```

MSDOS.SYS <- kernel

IO.SYS <- device drivers

---

2. Layered Structure:

- OS is divided into multiple layers, so that each layer depends on the lower layer and provide functionality to upper layer.

---

applications

---

system call APIs

---

system call implmenetation

Kernel Executive : File Mgr, Memory Mgr,
Process Mgr, Scheduler, Thread Mgr, etc.

IO Subsystem

Device Drivers

Hardware Abstraction Layer

- Windows, UNIX, etc.

3. Monolithic Kernel:

- Multiple kernel source files are compiled into single kernel binary image. Such kernels are "mono-lithic" kernels.
- Since all functionalities present in single binary image, execution is faster.
- If any functionality fails at runtime, entire kernel may crash.
- Any modification in any component of OS, needs recompilation of the entire OS.
- BSD Unix, Windows (ntoskrnl.exe), Linux (vmlinuz), etc.

4. Microkernel:

- Kernel is having minimal functionalities and remaining functionalities are implemented as independent processes called as "servers".
- e.g. File management is done by a program called as "file server".
- These servers communicate with each other using IPC mechanism (message passing) and hence execution is little slower.
- If any component fails at runtime, only that process is terminated and rest kernel may keep functioning.
- Any modification in any component need to recompile only that component.
- e.g. Symbian, MACH, etc.

5. Modular Kernel:

- Dynamically loadable modules (e.g. .dll / .so files) are loaded into calling process at runtime.
- In modular systems, kernel has minimal functionalities are rest of the functionalities are implemented as dynamically loadable modules.
- These modules get loaded into the kernel whenever they are called.
- As single kernel process is running, no need of IPC for the execution and thus improves performance of the system.
  - e.g. Windows, Linux, etc.

6. Hybrid Kernel:

- Mac OS X kernel is made by combination of two different kernels.
- BSD UNIX ✦ MACH = Darwin

✦ System Calls: [Dual Mode Protection]

- System calls are the functions exposed by the kernel so that user programs can access kernel functionalities.
- Kernel may have number of syscalls. E.g. UNIX OS have 64 syscalls.
- CPU have a mode bit indicating whether user program is running or system program is running.

  mode=0 => kernel/system/privileged mode

  mode=1 => user/non-privileged
- When interrupt occurs, mode is switched from user to kernel mode. And when interrupt is handled (ISR completes) mode is switched back from kernel to user mode.
- System call wrappers/APIs use software interrupt to switch to kernel mode. Then SW intr ISR select appropriate syscall implementation function from System call table and it is invoked.
- In kernel mode the complete access to the system is available and hence all kernel functions/device drivers can directly deal with hardware.
- If any user program try to access hardware directly, CPU will not execute those instructions (mode=1) and such program will be terminated forcefully.
- e.g. open(), close(), write(), read(), lseek(), fork(), exec(), _exit(), ...

✦ CPU Protection:

- To ensure that one process should not consume whole CPU time (e.g. if process stuck in infinite loop), timer hardware is used.
- Timer hardware is configured to generate interrupt periodically, which will cause execution of timer ISR.
- At the end of ISR, CPU scheduler is invoked, which decide the next task to be executed. And control (CPU time) is given to that process.

✦ Process Management:

- Process is program in execution.
- When a process makes system call, functions within kernel are executed. Function activation records of these kernel functions is created on the kernel stack of the process.
- Pointer to kernel stack is maintained in the PCB of the process.

✦ Context Switch:

- If interrupt occured while P1 is executing, values of all CPU registers (execution context) is copied into PCB of the P1 process.

- Then ISR is invoked, which will call scheduler and it will decide the next process to be executed

- Then dispatcher (another unit of OS) will copy the values of CPU registers from the PCB of P2 process into the CPU. Thus CPU will begin/resume execution of P2 process.

- This is called as "context switch".

- During context switch CPU cannot perform any useful task (as CPU registers are getting copied). So very freqent context switch will reduce CPU utilization.

- If context switch is done after long time, then system will not be responsive.

✦ **Process Life Cycle:**

✦ OS Data Structures:

- Job queue / Process table: PCBs of all processes in the system are maintained here.

- Ready queue: PCBs of all processes ready for the CPU execution and kept here.

- Waiting queue: Each IO device is associated with its waiting queue and processes waiting for that IO device will be kept in that queue.

- **Process states:**

- New, Ready, Running, Waiting, Terminated.

✦ **Inter-process Commmunication:**

✦ Shared Memory:

- A process can transfer data to another process using shared memory.

- It is a common memory area created by OS, in which multiple processes can read/write.

- To keep info about shared memory areas OS maintains structure i.e. shared memory object, which contains: KEY (uniqe id), size, permissions, address of shared memory, number of processes attached to shared memory, etc.

- Addresses of all shared memory objects are kept in a shared memory table and index to shared memory table is called as "shared memory id".

- Imp syscalls:

  - shmget() - create shm region
  - shmat() - get pointer to shm region
  - shmdt() - free pointer to shm region
  - shmctl() - to get info about shm or to delete shm.
  - Commands:

    ipcs -m

    ipcrm -m key

  - Shared memory is fastest IPC mechanism.

✦ **Pipe**

- Pipe is used to communicate betn two processes.
- It is stream based uni-directional communication
- Pipe is internally implemented as a kernel buffer, in which data can be written/read.
- If pipe (buffer) is empty, reading process will be blocked.
- If pipe (buffer) is full, writing process will be blocked.
- If reading process is terminated, writing process will receive SIGPIPE signal.

- There are two types of pipe:
  - **Unnamed Pipe:**
    - Used to communicate betn related processes.
      - e.g. who | wc
    - Internally pipe is created using pipe() syscall.
  - **Named pipe / FIFO:**
    - Used to communicate betn unrelated processes.
    - Internally creates an inode and directory entry for a special pipe file on the disk.
    - Created using mkfifo command or mkfifo() syscall.
    - Opened using open() syscall by reader and writer process.
    - write(), read(), close() operations can be performed on both types of pipe.

✦ **Message Queue / Mailbox:**

- Used to transfer packets of data from one process to another.
- It is bidirectional IPC mechanism.
- Internally OS maintains list of messages called as "message queue" or "mailbox".
- The info about msg que is stored in a object. It contains unique KEY, permissions, message list, number of messages in list, processes waiting for a message to receive (waiting queue).
- **IMP Syscalls:**
  - msgget() - create msg que
  - msgctl() - get info or destroy msg que
  - msgsnd() - send message into que
  - msgrcv() - receive message from que

✦ **Signals**

- OS have a set of predefined signals, which can be displayed using command

  kill -l
- A process can send signal to another process or OS can send signal to a process.
- kill command is used to send signal to another process, which internally use kill() syscall.
- pkill command is used to send signal to multiple processes/instances of the same program.

  pkill -SIG programname
- **IMP Signals:**

1. SIGINT: When CTRL✦C is pressed, INT signal is sent to the foreground process. If process does not handle it, process will be terminated.
2. SIGTERM: During system shutdown, OS send this signal to all processes. Process can handle this signal to close resources and get terminated. If process does not handle it, process will be terminated.
3. SIGKILL: During system shutdown, OS send this signal to all processes to forcefully kill them. Any process cannot handle this signal.
4. SIGSTOP: Pressing CTRL✦S, generate this signal which suspend the foreground process.
5. SIGCONT: Pressing CTRL✦Q, generate this signal which resume suspended the process.
6. SIGSEGV: If process access invalid memory address (dangling pointer), OS send this signal to process causing process to get terminated after giving an error message "Segmentation Fault".
7. SIGCHLD: When child process is terminated, this signal is sent to the parent process.

Prasanna Wadekar

- signal() syscall is used to install signal handler. When signal is received OS calls appropriate signal handler.

✦ **Sockets**

- Socket is defined as communication endpoint.
- Using socket one process can communicate with another process on same machine or different machine in the network.
- Socket = IP address ✦ PORT
- PORT number is 16-bit logical number (0-65535) to identify socket uniquely on a system.
- Two types: TCP sockets / UDP sockets

✦ **Remote Procedure Call**

✦ Used to call method from another process on the same machine or different machine in the netwok.

✦ Stub and Skeleton are helper objects which transfer data between server and client using sockets.

✦ **Process Creation:**

✦ System Calls:

- Windows : CreateProcess()
- Linux : clone() / fork()
- UNIX : fork()

✦ fork():

- Example:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        int ret;
        printf("start!\n");
        ret = fork();
        printf("return val : %d\n". ret);
        printf("end!\n");
        return 0;
}
```

- To execute certain task concurrently we can create a new process (using fork() on UNIX).
- fork() creates a new process by duplicating calling process. The new process is called as "child" and calling process is known as "parent".
- fork() returns 0 to child process and pid of the child process to the parent process.
- Even if child is copy of the parent process, after its creation it is independent of parent and both these processes will be scheduled sepeately by the scheduler.
- Based on CPU time given for each process, both processes will execute concurrently.
- In UNIX/Linux each process have some parent process.
- The first process (process 1) is "init" process, which is created by a hardcoded "process 0".
- If parent of any process is terminated, that child process is known as orphan process.
- The ownership of such orphan process will be taken by "init" process.
- Command: ps -e -o pid,ppid,cmd

- When a process terminates, its exit status is returned to the OS. OS store it into PCB of that process (so that its parent can read it).
- If process is terminated before its parent process and parent process is not reading its exit status, then even if process's memory/resources is released, its PCB will be maintained. This state is known as "zombie state".
- To avoid zombie state parent process should read exit status of the child process. It can be done using wait() syscall (which performs 3 steps):
  - pause execution parent until child process is terminated.
  - read exit status & other info from PCB of child process & return to parent process (as out param).
  - release PCB of the child process.

✦ **exec() syscall:**

- exec() syscall "loads a new program" in the calling process's memory (address space) and replaces the older one.
- There are six functions in the family of exec():
  - execl(), execlp(), execle(),
  - execv(), execvp(), execve()
- Example:

```
ret = fork();
if(ret==0)
{
        err = execl("/bin/ls", "ls", ..., NULL);
        if(err < 0)
        {
                perror("execl() failed");
                _exit(1);
        }
}
wait(&staus);
```

- If exec() succeed, it does not return (rather new program is executed).

✦ **Program/Command Execution:**

✦ Synchronous Execution: Parent waits for the child process to complete its execution. Internally calls wait function. By default shell executes each command synchronously.

e.g. ./kcalc

✦ Asynchronous Execution: Parent does not wait for the child process to complete its execution. Shell can execute a command asynchronously by suffixing & to the command.

e.g. ./kcalc &

✦ **CPU Scheduling:**

✦ There are four scenarios when CPU scheduler is invoked:

A. Running -> Terminated (exit)
B. Running -> Waiting (io request)
C. Running -> Ready (interrupt)
D. Waiting -> Ready (io completes)

✦ **Two Types of scheduling:**

1. **Non-Preemptive Scheduling:**
   - Scheduler is invoked only in case A & B (where process give up CPU on its own).
   - Also called as "co-operative scheduling".

2. **Preemptive Scheduling:** Scheduler is invoked in all above cases (Specially in case C & D, running process is forced to leave the CPU).

✦ **CPU Scheduling Criterias:**

- CPU Utilization: MAX : Server:90%, Workstation:60%
- Throughput: MAX : Amount of work completed in unit time.
- Waiting Time: MIN : Time spent by the process in the ready queue waiting for the CPU.
- Turn around Time: MIN : Time from the arrival of process to termination of the process.
- Response Time: MIN : Time from the arrival of the process to first CPU allocation time for that process.

✦ **CPU Scheduling Algos:**

✦ **FCFS**
- Process arrived first will be scheduled first.
- Non-preemptive scheduling
- Avg waiting time depends on the order of arrival of the process.

✦ **SJF**
- Process having minimum burst time will be executed first.
- This algo gives minimum waiting time.
- Can be implemented as Non-premptive as well as Preemptive
- Preemptive SJF is also known as "Shortest Remaining Time First".

✦ **Priority**
- Each process is associated with a number called as priority of the process. Lower is the number, higher is the priority.
- Can be implemented as Non-premptive as well as Preemptive
- Process having lower priority may not get sufficient CPU timer for execution; this is called as "Starvation".
- To resolve this priotity of such processes can be incremented periodically; this is called as "Aging".

✦ **Round Robin**
- Each process is given a small CPU time (called as time quantum) repeatedly.
- Pre-emptive algorithm
- This algo gives minimum response time.

✦ **Fair Share:**
- CPU time is divided into epoch times.
- Each process will get fair amount of CPU time in each epoch based on its priority [nice value].
- By default a process inherit nice value from the parent process. It can be altered programmatically

using nice() system call.

- Root user can change priority of the process using "nice" or "renice" command.
- Lower is the nice value, higher is the CPU time share given to the process.

✦ **RTOS Scheduling Algorithms:**

✦ **Rate Monotonic Scheduling:**

- Static priority based scheduling algorithm i.e. priorities of the processes are not changed at run time.
- Does not utilize CPU fully as number of processes increase.

✦ **Earliest Deadline First:**

- The priorities of processes are dynamically changed to complete the processes within deadline.

✦ **Proportional Share Scheduling:**

- Based on priority of the process, it is given some amount of CPU shares.

✦ **Multi-Level Queue:**

- Depending on nature of the processes, the ready queue is splitted into multiple sub-queues and each queue can have different scheduling algo. This is known as "multi-level queue".
- If a process is not getting sufficient CPU time, in its current queue, then it can be shifted into another queue by OS. This enhanced concept is known as "multi-level feedback queue".

✦ **Process Synchronization:**

- Multiple processes accessing same resource at the same time, is known as "race condition".
- When race condition occurs, resource may get corrupted (unexpected results).
- Peterson's problem, if two processes are trying to modify same variable at the same time, it can produce unexpected results.
- To resolve this problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- Well known sync objects are:
  - Semaphore, Mutex, Event, Critical section, Waitable Timer, Monitor

✦ **Semaphore:**

- Semaphore is a sync primitive given by OS.
- Internally semaphore is a counter. On semaphore two operations are supported:

1. wait operation: dec op: P :

- semaphore count is decremented by 1.
- if cnt < 0, then calling process is blocked.
- typically wait operation is performed before accessing the resource.

2. signal operation: inc op: V :

- semaphore count is incremented by 1.
- if one or more processes are blocked on the semaphore, then one of the process will be resumed.
- typically signal operation is performed after releasing the resource.

Q. If sema count = -n, how many processes are waiting on that semaphore?

Ans: "n" processes waiting

Q. If sema count = 5 and 3 P & 4 V operations are performed, then what will be final count of sema?

Ans: 5 - 3 ✦ 4 = 6

✦ There are two types of semaphores:

1. Counting Semaphore:

- Allows "n" number of processes to access resource at a time.

2. Binary Semaphore:

- Allows only 1 process to access resource at a time.

✦ **IMP Syscalls:**

- sem_init() : init sem with some count
- sem_destroy() : destroy sem
- sem_wait() : wait/dec op
- sem_post() : signal/inc op

✦ **Mutex:**

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- However mutex is more efficient than binary semaphore.
- Mutex can be unlocked by the same process/thread, which had locked it.
- IMP syscalls:
  - pthread_mutex_create() : create mutex
  - pthread_mutex_destroy() : destroy mutex
  - pthread_mutex_lock() : lock mutex
  - pthread_mutex_unlock() : unlock mutex

✦ **Deadlock**

✦ Deadlock occurs when four conditions/characteristics hold true at the same time:

1. No preemption: A resource should not be released until task is completed.
2. Mutual exclusion: Resources is not sharable.
3. Hold & Wait : Process holds a resource and wait for another resource.
4. Circular wait: Process P1 holds a resource needed for P2, P2 holds a resource needed for P3 and P3 holds a resource needed for P1.

✦ **Deadlock Prevention:**

- OS syscalls are designed so that at least one deadlock condition does not hold true.

✦ **Deadlock Avoidance:**

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.
- **Algorithms used for this are :**

1. Resource allocation graph: OS maintains graph of resources and processes. A cycle in graph indicate circular wait will occur. In this case OS can deny a resource to a process.

2. Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.

3. Safe state algorithm: OS maintains statistics of number of resources and number processes. Based on stats it decides whether giving resource to a process is safe or not (using a formula):

Max num of resources required < Num of resources ✦ Num of processes

- If condition is true, deadlock will never occur.
- If condition is false, deadlock may occur.

Q. A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____

Ans:

3 * 2 < 4 ✦ 3

6 < 7 -> TRUE => Deadlock will not occur

✦ **Multi-Threading:**

- Multiple parts/functions in a process can be executed concurrently using "threads". It is called as "thread based multi-tasking" or "multi-threading".
- When OS creates a new thread, a new stack is created for the thread within process memory and also a controlling block is created for the thread (TCB).
- Each thread is associated with a function called as "thread function" or "thread procedure". When thread is created, it begins execution of that function and when function completes, thread is terminated.
- TCB of a thread contains info required to execute thread, which includes:
  - Thread Id, Scheduling Info (state, sched algo), User Stack address, Execution context, Kernel stack, etc.
- In modern OS, process is like a container and threads are executed within processes. They are executable units.
- In modern OS, scheduler allocated CPU time to the threads (not the process).
- In modern OS, for each process at least one thread is created by the OS, which execute the main() function. This thread is called as "main" thread.
- Programmer can create additional threads using syscalls or library functions:
  - Windows: CreateThread()
  - Linux: clone()
- It is standard programming to create threads using libraries instead of syscalls.
- POSIX thread library is used to create thread on UNIX/Linux/Mac OS X.

```
pthread_t th;
ret = pthread_create(&th, NULL, func, NULL);
```

arg1: out param contains thread id.
arg2: thread attributes e.g. stack size, priority, scheduling policy
arg3: address of thread func void* func(void*);
arg4: argument of the thread func.

- Thread is a lightweight process.
  - When a thread is created a new TCB and stack is created; while other sections are shared from the parent process.
  - Threads can communicate with each other directly via global/dynamic variables (data/heap section). Their communication is much faster than IPC.
  - Context switch between two threads of the same process is much faster.
- If parent process is terminated (container), all threads in that process are terminated.

✦ **Threading Model:**

- Threads created by thread libraries are used to execute functions in user program. They are called as "user threads".
- Threads created by the syscalls (or internally into the kernel) are scheduled by kernel scheduler. They are called as "kernel threads".
- User threads are dependent on the kernel threads. Their dependency/relation (managed by thread library) is called as "threading model".
- There are four threading models:

1. Many to One:

- Many user threads depends on single kernel thread.
- If one of the user thread is blocked, remaining user threads cannot function.

2. Many to Many:

- Many user threads depend on equal or less number of kernel threads.
- If one of the user thread is blocked, other user thread keep executing (based on remaining kernel threads).

3. One To One:

- One user thread depends on one kernel thread.

4. Two Level Model:

- OS/Thread library supports both one to one and many to many model.

✦ **Memory Management:**

- In multi-programming OS, multiple programs are loaded in memory.
- RAM memory should be divided for multiple processes running concurrently.
- Memory Mgmt scheme used by any OS depends on the MMU hardware used in the machine.
- There are three memory mgmt schemes are available:
  1. Contiguous Allocation:
  2. Segmentation:
  3. Paging:

✦ **Contiguous Allocation:**

A. Fixed Partition:

- RAM is divided into fixed sized partitions.
- This method is easy to implement.
- Number of processes are limited to number of partitions.
- Size of process is limited to size of partition.
- If process is not utilizing entire partition allocated to it, the remaining memory is wasted. This is called as "internal fragmentation".

B. Dynamic Partition:

- Memory is allocated to each process as per its availability in the RAM. After allocation and deallocation of few processes, RAM will have few used slots and few free slots.
- OS keep track of free slots in form of a table.
- For any new process, OS use one of the following mechanism to allocate the free slot.

1. First Fit: Allocate first free slot which can accommodate the process.
2. Best Fit: Allocate that free slot to the process in which minimum free space will remain.

3. Worst Fit: Allocate that free slot to the process in whic maximum free space will remain.

- Statistically it is proven that First fit is faster algo; while best fit provides better memory utilization.

- Memory info (physical base address and size) of each process is stored in its PCB and will be loaded into MMU registers (base & limit) during context switch.

- CPU request virtual address (address of the process) and is converted into physical address by MMU as shown in diag.

- If invalid virtual address is requested by the CPU, process will be terminated.

- If amount of memory required for a process is available but not contiguous, then it is called as "external fragmentation".

- To resolve this problem, processes in memory can be shifted/moved so that max contiguous free space will be available. This is called as "compaction".

✦ Virtual Memory:

- The portion of the hard disk which is used by OS as an extension of RAM, is called as "virtual memory".

- If sufficient RAM is not available to execute a new program or grow existing process, then some of the inactive process is shifted from main memory (RAM), so that new program can execute in RAM (or existing process can grow).

- It is also called as "swap area" or "swap space".

- Shifting a process from RAM to swap area is called as "swap out" and shifting a process from swap to RAM is called as "swap in".

- In few OS, swap area is created in form of a partition. E.g. UNIX, Linux, ...

- In few OS, swap area is created in form of a file E.g. Windows (pagefile.sys), ...

- Virtual memory advantages:

  - Can execute more number of programs
  - Can execute bigger sized programs

✦ Memory Management (continued):

✦ Segmentation:

- Instead of allocating contiguous memory for the whole process, contiguous memory for each segment can be allocated. This scheme is known as "segmentation".

- Since process does not need contiguous memory for entire process, external fragmentation will be reduced.

- In this scheme, PCB is associated with a segment table which contains base and limit (size) of each segment of the process.

- During context switch these values will be loaded into MMU segment table.

- CPU request virtual address in form of segment address and offset address.

- Based on segment address appripriate base-limit pair from MMU is used to calculate physical address as shown in diag.

- MMU also contains STBR register which contains address of process's segment table in the RAM.

✦ Demand Segmentation:

- If virtual memory concept is used along with segmentation scheme, in case low memory, OS may swap out a segment of inactive process.

- When that process again start executing and ask for same segment (swapped out), the segment will be loaded back in the RAM. This is called as "demand segmentation".

- Each entry of the segment table contains base & limit of a segment. It also contains additional bits like segment permissions, valid bit, dirty bit, etc.

- If segment is present in main memory, its entry in seg table is said to be valid (v=1). If segment is swapped out, its entry in seg table is said to be invalid (v=0).

✦ **Paging:**

- RAM is divided into small equal sized partitions called as "frames" / "physical pages".
- Process is divided into small equal sized parts called as "pages" or "logical/virtual pages".
- page size = frame size.
- One page is allocated to one empty frame.
- OS keep track of free frames in form of a linked list.
- Each PCB is associated with a table storing mapping of page address to frame address. This table is called as "page table".
- During context switch this table is loaded into MMU.
- CPU requests a virtual address in form of page address and offset address. It will be converted into physical address as shown in diag.
- MMU also contains a PTBR, which keeps address of page table in RAM.
- If a page is not utilizing entire frame allocated to it (i.e. page contents are less than frame size), then it is called as "internal fragmentation".
- Frame size can be configured in the hardware. It can be 1KB, 2KB or 4KB. ...
- Typical Linux and Windows OS use page size = 4KB.

✦ **TLB (Translation Look-Aside Buffer) Cache:**

- TLB is high-speed associative cache memory used for address translation in paging MMU.
- TLB has limited entries (e.g. in P6 arch TLB has 32 entries) storing recently translated page address and frame address mappings.
- The page address given by CPU, will be compared at once with all the entries in TLB and corresponding frame address is found.
- If frame address is found (TLB hit), then it is used to calculate actual physical address in RAM (as shown in diag).
- If frame address is not found (TLB miss), then PTBR is used to access actual page table of the process in the RAM (associated with PCB). Then page-frame address mapping is copied into TLB and thus physical address is calculated.
- If CPU requests for the same page again, its address will be found in the TLB and translation will be faster.
- **Context switch can be handled in one of the following ways:**

1. During context switch TLB is flushed i.e. all entries used by previous process will be cleared. Thus new entries page addresses will not conflict with prev process page addresses.
2. TLB can store page addresses along with PID of the process. In this case there is no need to flush TLB during context switch. The comparison of the page address will not conflict with another process, because it is associated with PID of the process. This combination PID and Page Address is called as "ASID".

✦ **Two Level Paging:**

- Primary page table has number of entries and each entry point to the secondary page table page.
- Secondary page table has number of entries and each entry point to the frame allocated for the process.
- Virtual address requested by a process is 32 bits including

  p1 (10 bits) -> Primary page table index/addr

  p2 (10 bits) -> Secondary page table index/addr

  d (12 bits) -> Frame offset
- If frame size is 4KB, 12 bits are sufficient to speficy any offset in the frame. This will also ensure that "d" will not contain any invalid frame offset.

- If virtual address of a process is of 32 bits [p1|p2|d], then maximum address of the process can be
= 1024 * 1024 * 4 * 1024 = 4 GB.

✦ **Page Fault:**

- Each page table entry contains frame address, permissions, dirty bit, valid bit, etc.
- If page is present in main memory its page table entry is valid (valid bit = 1).
- If page is not present in main memory, its page table entry is not valid (valid bit = 0).
- This is possible due to one of the following reasons:
  - Page address is not valid (dangling pointer)
  - Page is swapped out.
- If CPU requests a page that is not present in main memory (i.e. page table entry valid bit=0), then "page fault" occurs.
- Then OS's page fault exception handler is invoked, which handles page faults as follows:

1. Check virtual address due to which page fault occured. If it is not valid (i.e. dangling pointer), terminate the program.
2. If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
3. If page is on swap device, swap in the page in that frame.
4. Update page table entry i.e. add new frame address and valid bit = 1.
5. Restart the instruction for which page fault occurred.

✦ **Page Replacement Algorithms:**

- While handling page fault if no empty frame found (step 2), then some page of any process need to be swapped out. This page is called as "victim" page.
- The algorithm used to decide the victim page is called as "page replacement algorithm".
- There are three important page replacement algos:

1. FIFO:

- The page brought in memory first, will be swapped out first.
- Sometimes in this algorithm, if number of frames are increased, number of page faults also increase.
- This abnormal behaviour is called as "Belady's Anomaly".

2. OPTIMAL:

- The page not required in near future is swapped out.
- This algorithm gives minimum number of page faults.
- This algorithm is not practically implementable.

3. LRU:

- The page which not used for longer duration will be swapped out.
- This algorithm is used in most OS like Linux, Windows, ...
- LRU mechanism is implemented using "stack based approach" or "counter based approach".

✦ **Thrashing:**

- If number of programs are running in comparatively smaller RAM, a lot of system time will be spent into page swapping (paging) activity.
- Due to this system performance is reduced.
- The problem can be solved by increasing RAM size in the machine.

✦ **Copy On Write:**

- fork() syscall creates a logical copy of the calling process.

- Initially, child and parent both processes share the same pages in the memory pages.

- When one of the process try to modify contents of a page, the page is copied first: so that parent and child both will have seperate physical copies of that page. This will avoid modification of a process by another process.

- This concept is known as "copy on write".

- The primary advantage of this mechanism is to speed up process creation (fork()).

✦ **Dirty Bit:**

- Each entry in page table has a dirty bit.
- When page is swapped in, dirty bit is set to 0.
- When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area.
- When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.
    - If bit=1 (page is modified), page is physically overwritten on its older copy in the swap area.

✦ **Demand Paging:**

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- These pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".

✦ **File Management:**

- File is collection of data/info on secondary storage device.
- Any file has
    - data (contents of the file)
    - meta-data (info about the file).
        - size of file
        - permissions of file
        - user/group info
        - time stamps : creation, accessed, modified
        - type of file
        - info about data blocks of that file.

✦ **Hard Disk:**

- Hard disk is a block device. The data can be read/written in the disk block by block.
- Number of bytes that be read/written on the disk at a time is called as "physical block size" (sector). Typically it is 512 bytes and is decided by the manufacturer.
- OS/Filesystem may read/write disk in terms of larger block size, which is called as "logical block size".
- Logical block size is always multiple of physical block size.

✦ **File System:**

- File system is way of organizing the contents on the disk.
- During formatting file system is created.

- Formatting commands:
    - Windows : format
    - UNIX/Linux : mkfs
- Any file system is logically divided into four parts:
    - Boot block / Boot sector:
        - Contains bootstrap and boot loader program is partition is bootable.
    - Super block / Volume Control block:
        - Contains info about the partition
        - Partition size, label
        - Number of data blocks, number of free data blocks and info about free data blocks.
    - i-node list / Mater file table:
        - The info/metadata of a file is stored in a structure called as "file control block" or "i-node".
        - Collection i-nodes in a partition is called as "i-node list".
    - Data blocks:
        - The data/contents of a file are stored in data blocks.

✦ **Types of File:**

- From (UNIX) OS perspective there are 7 types of files:
    - regular file
    d directory file
    l link file
    p pipe/fifo file
    s socket file
    c char device file
    b block device file

✦ **Directory:**

- From end user perspective, directory is a container which contains sub-directories and files.
- However, OS treats directory as a special file. The directory file contains one entry for each subdirectory or file in it.
- Each directory entry contains i-node number and name of sub-dir / file.

✦ **Links:**

- There are two types of links in Linux/UNIX.

1. Symbolic Link:

- ln -s /path/of/target/file linkpath
- Internally use symlink() syscall.
- A new link file is created (new inode and new data block is allocated), which contains info about the target file.
    - Link count is not incremented.
        - If target file is deleted, link becomes useless.
        - Can create symlinks for directories.

2. Hard Link:

- ln targetfilepath linkpath
- Internally use link() syscall.
- A new directory entry is created, which has a new name and same inode number. No new file (inode and data blocks) is created.
- Link count in the inode of the file is incremented.
    - If directory entry of target file is deleted (rm command), file can be still accessed by link directory entry.
    - Cannot create link for directories.

✦ **rm command:**

- The rm command in Linux, internally calls unlink() system call.
- It deletes directory entry of the file.
- It decrements link count in the inode by 1.
- If link count = 0, the inode is considered to be deleted/free. It can be reused for any new file.
- When inode is marked free, data blocks are also made free, so that they can also be reused for some new file.

✦ **File System Architecture:**

- Virtual File System:
    - This layer redirect file system request to the appropriate file system manager.
- File system manager:
    - File system manager enables access to repective file system on the disk.
    - OS can see all partitions whose file system managers are installed in that OS.
- IO subsystem:
    - Implement buffer cache and other mechanisms to speed up disk IO.

✦ **open() syscall:**

```
fd = open("/home/nilesh/abc.txt", O_RDONLY);
```

1. Convert given file path into its inode number. This is called as path name translation and is done by a kernel internal function namei().
2. Read inode of the file from the disk into inode table in memory. Inodes of all recently accessed files are kept in this table.
3. A file position is initialized to 0 and is stored in the open file table. Info of all files opened in the system, is maintained in this table.
4. Each process is associated with a file descriptor table. It keeps info of all files open by that process. For any process, three files are opened by default: 0-stdin, 1-stdout, 2-stderr. This entry stores mode in which file is opened and pointer to OFT entry.
5. Finally index to file desc table entry is returned, which is called as "file descriptor". All further read(), write(), lseek(), close() operations will be using this file desc.

✦ **Disk Allocation Mechanisms:**

Each file system allocate data blocks to the file in different ways:

1. Contiguous Allocation:

- Number of blocks required for the file are allocated contiguously.
- inode of the file contains starting block address and number of data blocks.

- Faster sequential and random access
- Number of blocks required for the file may not be available contiguously. This is called as "External Fragmentation".
- To solve this problem, data blocks of the files can be shifted/moved so that max contiguous free space will be available. This is called as "defragmentation".

2. Linked Allocation:

- Each data block of the file contains data/contents and address of next data block of that file.
- inode contains address of starting and ending data block.
- No external fragmentation. faster sequential access
- Slower random access.
- e.g. FAT

3. Indexed Allocation:

- A special data block contain addresses of data blocks of the file. This block is called as "index block".
- The address of index block is stored in the inode of the file.
- No external fragmentation. faster random and sequential access.
- File cannot grow beyond certain limit.

✦ **Free Space Management Mechanisms:**

- To allocate free blocks to any file. they should be located quickly. For this reason, info of free blocks is maintained in the super block using one of the following algorithm:

1. Bit Vector:

- Array of bits is used to keep info of used/free blocks.
- Number of bits = number of blocks.
- If nth bit=1, it means nth block is used.
- If nth bit=0, it means nth block is free.

2. Linked List:

- Super block contains address of first free block.
- Each free block contains address of next free block.

3. Grouping:

- One free block contains addresses of remaining free blocks.

4. Counting:

- Each entry in a free block keep starting free block address and number of free blocks after that.

✦ **Disk Scheduling:**

✦ **Hard disk structure:**

- Time required to perform read/write operation on particular sector of the disk, is called as "disk access time".
- Disk access time includes two components = seek time and rotational latency.
- Seek time is time required to move head to desired cylinder (track).
- Rotational latency is time required to rotate the platters so that desired sector is reached to the head.

✦ **Disk Scheduling Algorithms:**

- When number of requests are pending for accessing disk cylinders, magnetic head is moved using certain algo. They are called as "disk scheduling algorithms".

1. FCFS:

- Requests are handled in the order in which they arrived.

2. SSTF - Shortest Seek Time First:

- Request of nearest (to current position of magnetic head) cylinder is handled first.

3. SCAN or Elevator:

- Magnetic head keep moving from 0 to max cylinder and in reverse order continuously serving cylinder requests.

4. C-SCAN:

- Magnetic head keep moving from 0 to max cyliner serving the requests and then jump back to 0 directly.

5. LOOK:

- Implementation policy of SCAN or C-SCAN.
- If no requests pending magnetic head is stopped.

****

✦ **Linux File System:**

- Hierarchy:

/ (root of filesystem)

|- bin (programs and commands)
|- sbin (system programs and commands)
|- lib (library files used by various applns *.so)
|- boot (bootloader, kernel image & booting files)
|- home (for each user one directory is created here).
|- root (home dir of root user)
|- usr (user installed programs)
|- var (for system logs)
|- dev (files representing hw devices)
|- proc (info about the system)
|- sys (info about the devices)
|- etc (system config files)

- If username is "dac", then its home directory is "/home/dac". Any user must save all data/files into his/her home directory.
- The "root" user in Linux have full access to all system files. The "root" user's home directory is "/root".
- config files:
  - /etc/passwd (username and passwd info)
  - /etc/inittab (username and shell info)
  - /etc/fstab (info of mounted partitions)
- proc filesystem:
  - /proc/cpuinfo : complete hardware detail of CPU.

✦ **Linux Commands:**

- Linux is a multi-user system i.e. multiple users can login and work concurrently on the single Linux machine.
- If Linux is installed on your machine [SuSE Linux], then the terminals can be accessed using keys as follows:

  CTRL ✦ ALT ✦ F1 -> tty1

  to

  CTRL ✦ ALT ✦ F7 -> tty7 -> :0 -> GUI
- Within GUI terminal, multiple terminal windows can be opened, called as psuedo terminals. e.g. pts/0, pts/1, ...
- The current terminal name can be found using "tty" command.
- Command for knowing current user name: whoami
- Command for knowing current user, current terminal and login time is: who am i
- To know all users connected to the server machine: who
- To know detailed info about uses connected to server: w
- To close the terminal: exit

✦ **Path:**

✦ **absolute path:**

- full path of the file or directory
- always starts from "/" (root of file system)

✦ **relative path:**

- path of file or directory with respect to current directory
- does not start with "/"

✦ **Special Directory Symbols:**

.. -> represent parent directory

. -> represent current directory

~ -> represent user's home directory

✦ **Linux Basic Commands:**

1. pwd
2. mkdir dirpath
3. ls

   ls dirpath

   ls -l
4. cd dirpath
5. rmdir dirpath
6. cat > filepath

   ....

   ....

   CTRL + D
7. cat filepath
8. rm filepath
9. rm -r dirpath
10. cp command:

    cp filepath dirpath => copy given file in given dir

cp filepath destfilepath=> copy given file with another name

cp -r dirpath destdirpath=> copy given dir to given dest dir

11. mv command:

mv filepath dirpath => move given file into given dir

mv dirpath destdirpath => move given dir into given dest dir

mv filename newfilename => rename a file.

12. man commandname

13. touch filepath

- If file already exists, its "modification time" will be changed.
- If file does not exists, a new empty file is created.

14. echo command:

echo "to display a message"

echo -e "hi\tall\nhow was 1st day with linux?\ngood??"

echo -n "no newline after the text"

echo "can display variable value using $varname"

15. Environment variables keep important info about the system.

Command: env

16. grep -> GNU Reguar Expression Parser

- Used to search a pattern in single/multiple file(s).
- grep "word" filepath
- grep -R "word" dirpath
- egrep "fibonac+i" filepath

+ -> char is repeated 1 or more times

* -> char is repeated 0 or more times

? -> char is repeated 0 or 1 time

{n} -> char is repeated n times

{m,n} -> char is repeated minumum m and maximum n times

^ -> match at beginning of line

$ -> match at the end of line

[char1|char2] -> search a from options

[c1-c2] -> search a char from given range

. -> any one character

grep -> to search basic regular expression

egrep -> to search extended regular expression -> grep -E

fgrep -> to search a fixed word -> grep -F

17. chmod command:

- Used to give permissions to a file.
- chmod +x filepath
  - will add execute permission to all [ugo].
  - +x, +w, +r to add execute, write and read perms respectively.
- chmod -w filepath
  - will remove write permission from all [ugo].

- -x, -w, -r to remove execute, write and read perms respectively.
- chmod u+x filepath
  - will add execute permission for the user (owner of file)
- chmod g-w filepath
  - will remove write permission for the group.
- u+w, u+r, u+x, u-w, u-r, u-x
  g+w, g+r, g+x, g-w, g-r, g-x
  o+w, o+r, o+x, o-w, o-r, o-x
- to give permission to file octal form
  PERM -> rwx r-x r--
  111 101 100
  7 5 4
  chmod 754 filepath
- to change permission of all files in a directory
  chmod -R perm dirpath

18. chown command:
    - chown newuser filepath
    - chown newuser:group filepath
    - chown -R newuser:group dirpath

✦ **Redirection:**

1. output redirection
   command > filepath
   e.g. ls -l /home > output.txt
   command >> filepath
2. input redirection
   command < filepath
   e.g. wc < input.txt
   e.g. wc < input.txt > output.txt
3. error redirection
   command 2> filepath
   e.g. ls -l /ho 2> error.txt
   * command < input.txt > output.txt 2> error.txt

✦ **Pipe**

- Used to communicate between two processes.
- Output of a command can be redirected to another command.
- command1 | command2
- e.g. cat -n names.txt | head -11 | tail -7

✦ **VI editor**

- Developed by Bill Joy

- Editor works in two modes
    - Insert (Edit) mode : Press "i"
    - Command mode : Press "Esc"
- To open/create a file using VI editor

    vim filepath
- VI editor basic commands

    :w -> write

    :q -> quit

    :wq -> write and quit

    :q! -> quit without saving
- VI editor advanced command (to customize VI)

    :set number

    :set autoindent

    :set tabstop=4

    :set shiftwidth=4

    :set autowriteall
- VI editor copy/paste related commands:

    yy -> copy current line

    n yy -> copy n lines from cursor

    :m,ny -> copy from mth line to nth line

    p -> paste

    dd -> cut current line

    n dd -> cut n lines from cursor

    :m,nd -> cut from mth line to nth line

    u -> undo

    CTRL R -> redo

    y$ -> copy from cursor to end of line

    y^ -> copy from cursor to start of line

    yw -> copy current word

    n yw -> copy n words after the cursor

    d$ -> cut from cursor to end of line

    d^ -> cut from cursor to start of line

    dw -> cut current word

    n dw -> cut n words after the cursor

✦ **C Programming on Linux:**

step 1: vim filename.c

Write and save the C source code.

step 2: gcc filename.c -o filename.out

Compile .c file and create executable (.out) file.

step 3: ./filename.out

To run the program.

✦ **C++ programming on Linux:**

step 1: vim filename.cpp

Write and save the C++ source code.

step 2: g++ filename.cpp -o filename.out

Compile .cpp file and create executable (.out) file.

step 3: ./filename.out

To run the program.

✦ **C/C++ Program Debugging:**

- While compilation use -g flag

gcc filename.c -g -o filename.out

- While debugging

gdb ./filename.out

- Set break points wherever appropriate

gdb> break funname

gdb> break linenum

gdb> run

- Important gdb commands:

cont -> run till next break point

step -> step into the function

next -> step over the function

print varname -> watch variable value

quit -> stop debugging

help -> to see help about gdb commands

✦ **BASH Shell Scripts:**

- BASH reference manual - ebook

- #!/bin/bash -> shebang line
- comments starts with #

- do not use $ sign while assigning value to var

var=123

- to read value of var use $ sign

echo $var

newvar=$var

- to execute command and assign output to a variable

var=$(command ...)

var=`command ...`

- to execute command

command ...

- relational operator

```
        -eq -ne -lt -le -gt -ge
- logical operators
        -a -o !

-       if [ condition ]
        then
                ...
        fi

-       if [ condition ]
        then
                ....
        else
                ....
        fi
```

- Can use "elif" as combination as else-if.

```
-       case expression in
        c1)
                ...
                ;;
        c2)
                ...
                ;;
        *)
                ...
        esac

-       while [ condition ]
        do
                ....
        done
```

- break & continue can be used with all types of loops (while, until, for loop) like C/C++

```
-       until [ condition ]
        do
                ....
        done

-       for var in collection
        do
                ....
        done
```

The for loop is used to access elements one by one from the collection.

- Special Operators to use in a condition.

```
        -e check if path exists
        -d check if path is of directory
```

-f check if path is of file
-r check if path is readable
-w check if path is writable
-x check if path is executable

- All conditions in script [...] are executed internally with "test" command.
- man test

✦ **Positional Parameters:**

- Extra info passed to the script while executing it on command line is called as "positional parameters".
- ./scriptname arg1 arg2 arg3 arg4
- They can be accessed in script as : $1, $2, ..., $9
- Number of positional parameters : $#
- List of positional parameters : $*
- Name of current shell script : $0
- "shift n" command discard first n args and next arguments can be accessed
- Example:

```
#!/bin/bash
echo $1
echo $2
shift 2
echo $1
echo $2
```

- If above script runs as: ./script arg1 arg2 arg3 arg4

The output will be:

```
arg1
arg2
arg3
arg4
```

- Using shift command we can access parameter 10 and onwards.

- Functions:

```
function funname()
{
        # args are accessed as $1, $2, ...
        ....
}
res=$(funname ...)
```

- Functions must be defined at the start of the script
- All results of echo statements will be kept in temp string and will be returned at the end of the function, which can be collected in another variable as shown above.

Q.1 An OS uses a paging system with 1K pages. A given process uses a virtual address space of 128K and is assigned 16K of physical memory. How many entries does its page table contain?

Answer: 128

Explain:

virtual address space =	128KB	131072 Bytes
number of frames/pages =	1K	1024
size of one page =	virtual addr space / no of pages	128 Bytes
physical memory of process =	16KB	16384 Bytes
number of pages for process =	physical memory / page size	128
number of page table entries =	number of pages for process	128

Q.2 An OS uses the elevator algorithm to schedule the disk-arm. I/O requests are currently pending for blocks on tracks 1, 3, 8, 11, 15, and 16. The disk arm is currently at track 9 and moving upwards. In what order will these requests be handled?

Answer: 11, 15, 16, 8, 3, 1

Explain: elevator algorithm = SCAN algorithm

Q.3 A particular system uses a page size of 1K bytes. A page table for a particular process begins as follows: [3, 4, *, 1, *, 8 ...]

* A. What physical address corresponds to the virtual address of 50?
* B. Name a virtual address that will generate a page fault.

Answer: A: 3122

B: Any Addr betn 2048 to 3071 OR betn 4096 to 5119

Explain:

Logical Address	Physical Address
0	3072
1024	4096
2048	INVALID
3072	1024
4096	INVALID
5120	8192

Page Table

Logical Page	Physical Frame	Valid/Invalid
0	3	valid
1	4	valid
2	*	invalid
3	1	valid
4	*	invalid
5	8	valid

A: Address 50 = Page 0 address => Physical Frame 3 i.e. Frame 3 address + 50 (offset) => 3072 + 50 = 3122

B: Range of addresses for which physical addresses are not present : Any virtual address between 2048 and 3071 or between 4096 and 5119

Q.4 **With a segmentation, if there are 64 Segments and the maximum segment size is 512 words, the length of the logical address in bits is_____**

Answer: **16**

Explain: In segmentation scheme, Logical address contains "segment address" and "offset address".

64 segments :	segment address bits =	6
512 words = 512 * 2 = 1024 bytes :	offset address bits =	10
	Total Address Bits =	16

Q.5 **In an operating system using paging , if each 32-bit address is viewed as a 20-bit page identifier plus aS 12 bit offset, what is the size of each page?**

Answer: **4KB**

Explain: Page offset = 12 bits

So Page size =	2 ^ 12	4096 bytes

Q.6 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The largest allocation request in Kbytes that could be denied is for _____.**

Answer: **541 KB**

Explain: Assuming that both allocations are done contiguously at the bottom or top of the memory, the rest of the memory will be empty

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB

So cannot allocate the block greater than it...

Q.7 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for _____.**

Answer: **181 KB**

Explain: For the smallest allocation of the memory, we need to assume that memory is allocated in the middle as follows so that X, Y, Z almost equal ...

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB

Divide remaining memory so that X, Y, Z will be equal i.e. Remaining Memory Divided by 3

Smallest Available Slot =	540 KB / 3	180 KB

So we cannot allocate any slot larger than it

X
200
Y
260
Z

Q.8 If Semaphore values have 12, then 6V and 4P operations the resultant value is _____

Answer: 14

Explain: Each P operation decrements semaphore count by 1
Each V operation increments semaphore count by 1
Original Count = 12
Incrementing 6 Times = 18
Decrementing 4 Times = 14
Final Count = 14

Q.9 **Disk request come to a disk driver for cylinders in the order 10,22,20,2,40,6 and 38, at time when the disk drive is reading from cylinder 20. The seek time is 6 ms per cylinder. the total seek time, if the disk arm sheduling algorithm is first come first served is _____.**

Answer: **876 ms**

Explain: Seek Sequence for FCFS will be :
20, 10, 22, 20, 2, 40, 6, 38

		20	10
		10	12
		22	2
		20	18
		2	38
Total Seek Distance =	146	40	34
Seek time per cylinder =	6 ms	6	32
Total Seek Time =	876 ms	38	
		Total Seek Dist :	146

Q.10 Each process p(i)=1,2,3,.....,9 is coded as follows
repeat
p(mutex)
{critical section}
v{mutex}
forever
The code for p10 is identical except that it uses V(mutex) instated of p(mutex).What is the largest no of processes that can be the inside the critical section at any moment?

Answer: 3

Default behaviour of P() / wait operation locks mutex, while V() / signal operation unlocks mutex.
Due to this nature, only one process out of P1 to P9, will be inside the mutex.(Now there is 1 process)
The P10 process calls V() operation and enters into critical section (Now threre are 2 processes)
However, since P10 unlock the mutex, one more process from P1 to P9 can enter into critical section. (Now threre are 2 processes)
Thus there can be at max 3 processes in the critical section.

Q.11 **At a perticular time of computation the value of counting semaphore is 7. Then 20 `p` operation and x `v` operation were completed on this semaphore. If the final value of the semaphore is 5, x will be _____**

Answer : **18**

P() operation decrement count, while V() operation increment the count.

Current Count =	7
P() Operations =	20
New Count =	-13
Final Count =	5
V Operations =	18

Q.12 **A semaphore count of negative n means (s=-n) that the queue contains _____ waiting processes.**

Answer : n

If semaphore count is zero and any process perform decrement operation the process goes to waiting state,
Thus the first process doing decrement operation will be waiting and semaphore count -1.
So n waiting processes will make semaphore count = -n

Q.13 **The OS uses a round robin scheduler. The FIFO queue of ready processes holds three processes A,B,C in that order. The time quantum is 18 msec. A context switch takes 2 msec. After running for 13 msec, B will block to do a disk read, which will take 30 msec to complete. Trace what will happen over the first 100 msec. What is the CPU efficiency over the first 100 msec?**

Answer : **90%**

A FIFO Queue indicate that processes are added at the end of the queue and removed from the beginning (it is not FCFS algo).
During context switch CPU is not doing some fruitful work and hence is not utilized at that time.
When a process goes for IO, it is removed from READY queue and is added to IO WAITING queue of particular device.
Considering all these points the seqence of scheduling will be as follows (including context switch):
A (18) => SW (2) => B (13 io) => SW (2) => C (18) => SW (2) => A (18) => SW (2) => C (18) => SW (2) => B (5)

Total Milliseconds =	100 ms
Unused time (switching) =	10 ms
CPU utilization / efficiency =	90 ms
CPU utilization =	90 %

Prasanna Wadekar

Q.14 In a page memory the page hit ratio is 0.35. The time required to access the page in secondary memory is equal to 100 nanosec. the time required to access page in primary memory is 10 ns . the avg time reqd to access page is _____.

Answer : 68.5 ns

Page hit ratio represent probability of getting the page in main memory.

Hit ratio = 0.35 i.e. out of 100, only 35 pages will be available in main memory.

i.e. out of 100, only 65 pages will be available taken from secondary memory.

Time for accessing page in memory = 10 ns

Total time for accessing 35 pages = 350 ns

Time for accessing page in disk = 100 ns

Total time for accessing 65 pages = 6500 ns

Total time for accessing 100 pages = 6850 ns

Average time for accessing 1 page = 68.5 ns

Q.15 A 1000 MB hard disk has 512-byte sectors. Each track on the disk has 1000 sectors. The number of tracks on the disk is _____.

Answer : 2048

disk size = number of tracks * sectors per track * sizeof one sector

number of tracks = disk size / (sectors per track * sizeof one sector)

number of tracks = 1000 MB / (1000 * 512)

2048

Q.16 **The address sequence generated by tracing a particular program executing in a pure demand paging system with 100 records per page, with 1 free main memory frame is recorded as follows. what is the number of page faults?**

0100, 0200, 0430, 0499, 0510, 0530,0560, 0120, 0220, 0240, 0260, 0320,0370

Answer : **7**

Here no of records per page is 100, so the page no corresponding to these address will be

1,2,4,4,5,5,5,1,2,2,2,3,3

Also only one frame is available along with pure demand paging

So, Page fault will occur for frame nos. 1,2,4,5,1,2,3

Hence no of page fault is =7

Q.17 **Peak bandwidth of 64 bits, 33 MHZ based PCI bus would be _____**

Answer : **266 MB / sec**

Peak Bandwidth =	Bus Width (in Bytes) * Bus Freq (in Hz)	
Bus Width =	64 Bits	8 Bytes
Bus Freqnecy =	33 MHz	33.3 MHz
Peak Bandwidth =		266.4 MB/sec

Q.18 **Cosider a system having m resourses of same type. The resourse are shared by 3 processes A,B,C which have peak time demands of 3,4,6 respectively. The minimum value of m that ensures that deadlock will never occur is _____.**

Answer : **11**

Formula for resources of the same type :
Total of max needs < no. of resource instances + no. of processes
13 < m + 3
So, m >= 11

Q.19 **A certain moving arm disk storage with one head has following specification:**
Number of tracks/recording surface = 200
Disk rotation speed = 2400 rpm
Track storage capacity = 62500bits
the avg latency time (assume that the head can move from one track to another only by traversing the entire block)

Answer : **2.5 sec**

	Number of tracks =	200	
	Number of rev per sec =	2400 / 60	40
	Time required for 1 revolution =	1 / 40	0.025 sec
Correct	Average Latency =	0.5 * time for 1 rev	0.0125 sec
But Ans	Average rotational latency =	0.5 * tracks * time for 1 rev	2.5 sec

Track storage capacity = 62500 bits
Data transfer rate = number of rev (tracks) per sec * track storage capacity
i.e. 40 * 62500 bits / sec
2500000 bits/sec
2.5 mb/sec

Q.20 A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____

Answer: Can Never Occur

Formula for resources of the same type :
Total of max needs < no. of resource instances + no. of processes
6 < 4 + 3
This holds true, indicating that deadlock never occur

Q.21 In a multi-user OS 20 requests are made to particular resource per hour on an avg, the probability that no request are made in 45 minute is ___

Answer: e ^ -20

Avg requests per unit time (1 hr) =			20
Num of req per 45 mins i.e. L =			15
Probability is given by =	(L ^ x * e ^ -L) / x!		
x is num of expected reqs i.e. 0			
So probability will be =	(15 ^ 0 * e ^ -15) / 0!	=	e ^ -15

Prasanna Wadekar 9870248407

19:00

SUNBEAM

Institute of Information Technology

Placement Initiative

Data Structure:

- Structure of Data
  - The organization of data in memory.
  - Operations performed on that data.
- Types of data structures:
  - Linear - Array, Stack, Queue, Linked List.
  - Non-Linear - Tree, Graph
  - Associative:- data is stored in key value pair e.g. HashTable
- Usually data structures are represented as "Abstract Data Types".
- Example Data Structures:
  - Array, Stack, Queue, Linked List, Tree, Graph, Hashtable, etc.

Stack (ADT):

- Operations:
  - Push()
  - Pop()
  - Peek()
  - IsEmpty()
  - IsFull()
- Data Organization In Memory:
  - Using Arrays:
    - int arr[MAX];
    - int top;
  - Using Linked List;
    - node *head;
    - Push() ==> AddFirst();
    - Pop() ==> DelFirst();
    - Peek() ==> return head->data;
    - IsEmpty() ==> return head==NULL;

PRASANNA

Array (ADT)

- Operations:
  - GetAt(index);
  - SetAt(index, value);
  - Sort();

- Selection Sort
- Bubble Sort
- Insertion Sort
- Quick Sort
- Merge Sort
- Heap Sort

- Search(key);
  - Linear Search
  - Binary Search

Time Complexity:

- Approximate measure of time required for completing any algo/operation.
- Example: Fatorial:

```
res = 1;
for(i=1; i <= num; i++)
    res = res * i;
printf("factorial : %d\n", res);
```

- Time complexity of any algo depends on number of iterations in that algo.
- In above program, time is proportional to number whose factorial is to be calculated.
- T α n . Hence time complexity is O(n) -> Order of n.
- If time required for any algo is constant (not dependent on any factor) i.e. T = k. Then time complexity is represented as O(1).
- Time complexity is represented in "Big O" notation.

Selection Sort:

```
for(i=0; i<n-1; i++)
{
    for(j=i+1; j<n; j++)
    {
        if(a[i] > a[j])
            swap(a[i], a[j]);
    }
}
```

- Number of iterations = 1 + 2 + 3 + ... + n-1 = n (n-1) / 2
- T α n (n-1) / 2
- T α n^2 - n

- If n >> 1, then n^2 >>>> n => Hence all lower order terms can be ignored.
- $T \alpha n^2$
- Time complexity = $O(n^2)$

Bubble Sort:

```
for(i=0; i<n-1; i++)
{
    for(j=0; j<n-1; j++)
    {
        if(a[j] > a[j+1])
            swap(a[j], a[j+1]);
    }
}
```

- Number of iterations = (n-1)(n-1)
- $T \alpha n^2 - 2n + 1$
- If n >> 1, then n^2 >>>> n => Hence all lower order terms can be ignored.
- $T \alpha n^2$
- Time complexity = $O(n^2)$

Linear Search:

```
for(i=0; i<n; i++)
{
    if(key==a[i])
        return i; // element found at index i
}
return -1; // ele not found
```

- Best case : "key" is found at index 0.
  - Number of iterations = 1
  - T = k
  - O(1)
- Worst case : "key" is found at last index (n-1).
  - Number of iterations = n
  - $T \alpha n$
  - O(n)
- Average case : "key" is found in middle.
  - Number of iterations (average) = n / 2
  - $T \alpha n / 2$
  - $T \alpha n$

- O(n)

Binary Search:

```
while(left <= right)
{
    mid = (left+right) / 2;
    if(key == arr[mid])
        return mid; // ele found at index "mid"
    if(key < arr[mid])
        right = mid - 1;
    else
        left = mid + 1;
}
return -1;
```

- Array must be sorted to use binary search.
- $2^i = n$ -> where 'n' is number of elements, 'i' is number of iterations
- i log 2 = log n
- i = log n / log 2
- T α log n / log 2
- T α log n
- Time complexity : O (log n)

Most common time complexities:

- O(1)
- O(log n)
- O(n)
- O(n log n)
- $O(n^2)$

Time complexities of different algorithms:

1. Stack push & pop: T = k. O(1)
2. Queue push & pop: T = k. O(1)
3. Singly Linked list with head pointer:
   a. Add first -> T = k. O(1)
   b. Add last -> T α n O(n)
   c. Del first -> T = k. O(1)
   d. Search -> T α n / 2 O(n)
4. Singly Linked list with head & tail pointer:
   a. Add first -> T = k. O(1)

b. Add last -> T = k. O(1)
c. Del first -> T = k. O(1)
d. Search -> T α n / 2 O(n)

Insertion Sort:

```
for(i=1; i<n; i++)
{
    temp = arr[i];
    for(j=i-1; j >= 0 && arr[j] > temp; j--)
       arr[j+1] = arr[j];
    arr[j+1] = temp;
}
```

- Time complexity:
  - Best case: O(n)
  - Average/worst case: $O(n^2)$

Recursion:

- Function calling itself is called as "recursive function".
- From programming perspective two things are important:
  - Explain process in terms of itself.
  - Terminating condition.
- Advantages:
  - Simplifies / reduce the code.
- Disadvantages:
  - Increased space complexity (function activation records are created per call).
  - Increased time (time required to create activation record and call the function).

Quick Sort:

```
void quick_sort(int arr[], int left, int right)
{
    // 0. if part contains single ele (left==right) or part is invalid
(left>right), stop.
    if(left>=right)
        return;
    // arr[left] is pivot
    int i = left, j = right;
    while(i < j)
```

```
    {
        // 1. from left find number greater than pivot - i index
        while(i <= right && arr[i] <= arr[left])
            i++;
        // 2. from right find number less than or equal to pivot - j index
        while(arr[j] > arr[left])
            j--;
        // 3. if i < j, swap arr[i] & arr[j]
        if(i < j)
            swap(arr[i], arr[j]);
    } // 4. repeat steps 1-3, till i < j

    // 5. swap pivot (arr[left]) and arr[j]
    swap(arr[left], arr[j]);
    // 6. quick_sort for left part -> 'left' to 'j-1'
    quick_sort(arr, left, j-1);
    // 7. quick_sort for right part -> 'j+1' to 'right'
    quick_sort(arr, j+1, right);
}
```

- Time Complexity :
  - Avg case : O(n log n)
  - Worst case : O(n^2)
- Pivot selection:
  - leftmost or rightmost element
  - middle element
  - average of all values
  - median of all values

Merge Sort

```
void merge_sort(int arr[], int left, int right)
{
    // 0. if part contains single ele (left==right) or part is invalid
(left>right), stop.
    if(left>=right)
        return;
    // 1. divide array in two equal partitions.
    int mid = (left + right) / 2;
    // 2. sort left part -> left to mid
    merge_sort(arr, left, mid);
    // 3. sort right part -> mid+1 to right
    merge_sort(arr, mid+1, right);
    // 4. allocate a temp array : num of elements (right - left + 1)
```

```
    int cnt = right - left + 1;
    int *temp = new int[cnt];
    // 5. merge two sorted partitions into this temp array
    int i=left, j=mid+1, k=0;
    while(i<=mid && j<=right)
    {
        if(arr[i] < arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }
    while(i<=mid)
        temp[k++] = arr[i++];
    while(j<=right)
        temp[k++] = arr[j++];
    // 6. copy temp array back to original array partition
    for(k=0; k<cnt; k++)
        arr[left+k] = temp[k];
    // 7. release temp array
    delete[] temp;
}
```

- Time Complexity :
  - Avg/Worst case : O(n log n)

Heap Sort:

- Array implementation of almost complete binary tree is called as "Heap".
- Time Complexity :
  - Avg/Worst case : O(n log n)

Queue (ADT)

- Operations:
  - Push()
  - Pop()
  - Peek()
  - IsEmpty()
  - IsFull()
- Data orgranization in memory:
  - Using linked list (with head & tail pointer):
    - front => node *head;
    - rear => node *tail;
    - Push() => AddLast()

- Pop() => DelFirst()
- Peek() => return head->data;
- IsEmpty() => return head==NULL;

- Using arrays:
  - int arr[MAX];
  - int front;
  - int rear;

Linear Queue:

```
1. Init()
   front = rear = -1;
2. Push()
   rear ++;
   arr[rear] = ele;
3. Pop()
   front ++;
4. Peek()
   return arr[front+1];
5. IsEmpty()
   front == rear
6. IsFull()
   rear==MAX-1
```

Circular Queue:

- In linear queue, when rear reaches MAX-1, even if few locations are vacant, it shows queue is full. In other words, in linear queue there is no proper utilization of memory.
- To overcome this limitation, circular queue is implemented.
- In circular queue, front & rear are incremented in circular fashion i.e. 0, 1, 2, ..., MAX-1, 0, 1, ...

Implementation 1

```
1. Init()
   front = rear = -1;
2. Push()
   rear = (rear + 1) % MAX;
   arr[rear] = ele;
3. Pop()
   front = (front + 1) % MAX;
   if(front==rear)
        front = rear = -1;
4. Peek()
   temp = (front + 1) % MAX;
   return arr[temp];
5. IsEmpty()
```

```
   front == rear && front == -1
6. IsFull()
   (front==-1 && rear==MAX-1) || (front==rear && front!=-1)
```

Implementation 2 – Using count

```
1. Init()
   front = rear = -1;
   count = 0;
2. Push()
   rear = (rear + 1) % MAX;
   arr[rear] = ele;
   count++;
3. Pop()
   front = (front + 1) % MAX;
   count--;
4. Peek()
   temp = (front + 1) % MAX;
   return arr[temp];
5. IsEmpty()
   count == 0
6. IsFull()
   count == MAX
```

Types of Queues:

1. Linear Queue:
   a. No proper utilization of memory.
2. Circular Queue:
   a. Proper utilization of memory.
   b. Increments rear and front in circular fashion i.e. 0, 1, 2, 3, ..., MAX-1, 0, 1, ...
3. Priority Queue:
   a. Each element is associated with some priority.
   b. Element with highest priority is popped out first. (No FIFO behavior).
   c. Typically it is implemented as sorted linked list. While insertion element is added at appropriate position as per its priority. So insertion time complexity will be O(n). While deleting first (head) element is deleted (as it is element with highest priority).
4. Double Ended Queue (deque):
   a. Can push/pop elements from both sides i.e. front as well as rear.
   b. Usually implemented as doubly linked list with head & tail. because time complexity at both end complexity is O(1)

Stack Vs Queue:

	Stack	Queue
1	Stack is LIFO utility data structure.	Queue is FIFO utility data structure.
2	Push and Pop operations are done from the same end i.e. "top".	Push and Pop operations are done from different ends i.e. "rear" and "front" respectively.
3	There are no types of stack. However in single array you can implement two or more stacks (called as multi-stack approach).	There are four types of queues i.e. linear, circular, priority and deque.
4	Applications: • Function activation records are created on the stack for each function call. • To solve infix expression by converting to prefix or postfix. • Parenthesis balancing • To implement algos like Depth First Search.	Applications: • Printer maintains queue of documents to be printed. • OS uses queues for many functionalities: Ready queue, Waiting queue, Message queue. • To implement algos like Breadth First Search.

STL: Standard Template Library:

`#include <stack>`
`stack<char> s;`
`s.push('A');`
`s.pop();`
`ele = s.top();  // s.peek()`
`s.empty();  // s.isempty()`
	`#include <queue>`
	`queue<char> q;`
	`q.push('A');`
	`q.pop();`
	`ele = q.front();  // q.peek()`
	`q.empty();  // q.isempty()`

Infix, Prefix, Postfix:

- These are notations to represent math equations.
  - Infix: A + B -> For human
  - Prefix: + A B -> For computers
  - Postfix: A B + -> For computers
- Infix expression:
  - 4 + 3 - (9 - 5) / 2 * 8 - 6
- To convert Infix to Prefix/Postfix considers the priorities:
  1. ( )
  2. $ or ^ i.e. Exponential Operator
  3. * / % (left to right)
  4. + - (left to right)

Infix to Postfix using stack:

- Traverse Infix string element by element from left to right.

- If operand is found, then append to the postfix string.
- If operator is found, then push it on the stack.
- If priority of topmost of stack is greater or equal to priority of current operator, pop it from the stack and append to postfix string. Repeat this step if required.
- When infix string is completed, pop all operators from the stack and append to postfix string (one by one).
- If opening '(' is found, then push it on the stack.
- If closing ')' is found, pop all operators from the stack and append to postfix string (one by one) until '(' is found on stack. Also pop and discard that '('.
- Example: 4 + 3 - (9 - 5) / 2 * 8 - 6

Symbol	Postfix	Stack
4	4	
+	4	+
3	4 3	+
-	4 3 +	-
(	4 3 +	- (
9	4 3 + 9	- (
-	4 3 + 9	- (-
5	4 3 + 9 5	- (-
)	4 3 + 9 5 -	-
/	4 3 + 9 5 -	- /
2	4 3 + 9 5 - 2	- /
*	4 3 + 9 5 - 2 /	- *
8	4 3 + 9 5 - 2 / 8	- *
-	4 3 + 9 5 - 2 / 8 * -	-
6	4 3 + 9 5 - 2 / 8 * - 6	-
	4 3 + 9 5 - 2 / 8 * - 6 -	

Infix to Prefix using stack:

1. Traverse Infix string element by element from right to left.
2. If operand is found, then append to the prefix string.
3. If operator is found, then push it on the stack.
4. If priority of topmost of stack is greater than priority of current operator, pop it from the stack and append to prefix string. Repeat this step if required.
5. When infix string is completed, pop all operators from the stack and append to prefix string (one by one).
6. If closing ')' is found, then push it on the stack.
7. If opening '(' is found, pop all operators from the stack and append to prefix string (one by one) until ')' is found on stack. Also pop and discard that ')'.
8. Reverse prefix string.

Postfix Evaluation:

1. Traverse postfix string from left to right.

2. If operand is found, push it on the stack.
3. If operator is found, pop two operands from the stack; calculate result and push it on stack.
   a. First popped is second operand; while second popped is first operand.
4. Repeat steps 1-2, until postfix string is completed.
5. At the end, pop final result from the stack.
6. Example: 4 3 + 9 5 - 2 / 8 * - 6 -

Symbol	Stack	Operation
4	4	
3	4 3	
+	7	4 + 3 = 7
9	7 9	
5	7 9 5	
-	7 4	9 - 5 = 4
2	7 4 2	
/	7 2	4 / 2 = 2
8	7 2 8	
*	7 16	2 * 8 = 16
-	-9	7 - 16 = -9
6	-9 6	
-	-15	-9 - 6 = -15

Prefix Evaluation:

1. Traverse prefix string from right to left.
2. If operand is found, push it on the stack.
3. If operator is found, pop two operands from the stack; calculate result and push it on stack.
4. First popped is first operand; while second popped is second operand.
5. Repeat steps 1-2, until prefix string is completed.
6. At the end, pop final result from the stack.

Linked List:

- **Terminologies:**
  - Linked list is a linear data structure that contains multiple records linked to each other.
  - Each item in the list is called as "Node".
  - Each node contains data and pointer (address) to the next node.
  - Typically linked lists are implemented as self-referential structure/class. The structure/class contains pointer of the same type to hold address of next node.
- **LinkedList (ADT):**
  - The commonly supported operations in a linked list is called as:
    A. AddFirst()

B. AddLast()
C. InsertAtPos()
D. Search()
E. DeleteFirst()
F. DeleteLast()
G. DeleteAtPos()
H. Clear()
I. Sort()
J. Reverse()

- **Linked list in C++:**

```
class list;
class node
{
  private:
      int data;
      node *next;
  public:
      node(int val=0);
      friend class list;
};
class list
{
   private:
      node *next;
   public:
      list();
      ~list();
      addfirst(int val);
      addlast(int val);
      insert(int val, int pos);
      void delfirst();
      void dellast();
      void del(int pos);
      node* search(int key);
      void sort();
      void reverse();
      void clear();
};
```

- **Types of Linked Lists:**
  - **Singly Linear Linked List**
    - **A. Singly Linked List – only head pointer:**
      - **1. Add First: O(1)**

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
      head = newnode;
else
{
      // newnode's next --> head
      newnode->next = head;
      // head --> newnode
      head = newnode;
}
```

2. Add Last: O(n)

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
      head = newnode;
else
{
      // traverse till last node
      trav = head;
      while(trav->next!=NULL)
            trav = trav->next;
      // trav's next --> newnode
      trav->next = newnode;
}
```

3. Insert at position: O(n)

```
// if list is empty or first pos, newnode is first node.
if(head==NULL || pos==1)
      addfirst(val);
else
{
      // create and initialize node
      newnode = new node(val);
      // traverse till pos-1 node
      trav = head;
      for(i=1; i<pos-1; i++)
            trav = trav->next;
      // newnode's next --> trav's next
      newnode->next = trav->next;
      // trav's next --> newnode
      trav->next = newnode;
```

```
}
```

4. Delete First: O(1)

```
if(head!=NULL)
{
        // keep address of first node in temp pointer
        temp = head;
        // take head to next node.
        head = head->next;
        // delete temp node.
        delete temp;
}
```

5. Delete At Position: O(n)

```
// if node to be deleted is first or list is empty
if(pos==1 || head==NULL)
        delfirst();
else
{
        // traverse till pos-1
        trav = head;
        for(i=1; i<pos-1; i++)
                trav = trav->next;
        // temp should point to the node to be deleted.
        temp = trav->next;
        // trav's next --> temp's next
        trav->next = temp->next;
        // delete temp node
        delete temp;
}
```

6. Delete All: O(n)

```
// delete all nodes one by one, until list become empty
while(head!=NULL)
        delfirst();
```

7. Traverse Linked List:

```
// start traversing from first node
trav = head;
while(trav!=NULL)
{
        // visit current node
        printf("%d, ", trav->data);
        // go to the next node
```

```
        trav = trav->next;
} // repeat until end of list is reached
```

B. Singly Linked List –head and tail pointer:

1. Add First: O(1)

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
        head = tail = newnode;
else
{
        // newnode's next --> head
        newnode->next = head;
        // head --> newnode
        head = newnode;
}
```

2. Add Last: O(1)

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
        head = tail = newnode;
else
{
        // tail's next --> newnode
        tail->next = newnode;
        // tail --> newnode
        tail = newnode;
}
```

3. Delete First: O(1)

```
if(head!=NULL)
{
        // keep address of first node in temp pointer
        temp = head;
        // take head to next node.
        head = head->next;
        // if deleted node was last node, then
        if(head==NULL)
                tail = NULL;
        // delete temp node.
        delete temp;
```

```
}
```

- **Singly Circular Linked List:**

  Last node's next pointer keeps address of first (head) node.

1. **Add First: O(n)**

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
{
      head = newnode;
      newnode->next = newnode;
}
else
{
      // traverse till last node
      trav = head;
      while(trav->next!=head)
            trav = trav->next;
      // trav's next --> newnode
      trav->next = newnode;
      // newnode's next --> head
      newnode->next = head;
      // head --> newnode
      head = newnode;
}
```

2. **Add Last: O(n)**

```
// if list is empty, newnode is first node.
if(head==NULL)
{
      head = newnode;
      newnode->next = newnode;
}
else
{
      // traverse till last node
      trav = head;
      while(trav->next!=head)
            trav = trav->next;
      // trav's next --> newnode
      trav->next = newnode;
      // newnode's next --> head
      newnode->next = head;
```

```
}
```

3. Insert at position: O(n)

```
// if list is empty or first pos, newnode is first node.
if(head==NULL || pos==1)
      addfirst(val);
else
{
      // create and initialize node
      newnode = new node(val);
      // traverse till pos-1 node
      trav = head;
      for(i=1; i<pos-1; i++)
            trav = trav->next;
      // newnode's next --> trav's next
      newnode->next = trav->next;
      // trav's next --> newnode
      trav->next = newnode;
}
```

4. Delete First: O(n)

```
// if head is null, return
if(head==NULL)
      return;
// if there is a single node, delete it
else if(head==head->next)
{
      delete head;
      head = NULL;
}
// if there are more than one node
else
{
      // keep address of head node in temp pointer
      temp = head;
      // traverse till last node
      trav = head;
      while(trav->next!=head)
            trav = trav->next;
      // take head to next node.
      head = head->next;
      // update last node's next --> new head
      trav->next = head;
      // delete temp node.
      delete temp;
}
```

5. **Delete At Position: O(n)**

```
// if node to be deleted is first or list is empty
if(pos==1 || head==NULL)
      delfirst();
else
{
      // traverse till pos-1
      trav = head;
      for(i=1; i<pos-1; i++)
            trav = trav->next;
      // temp should point to the node to be deleted.
      temp = trav->next;
      // trav's next --> temp's next
      trav->next = temp->next;
      // delete temp node
      delete temp;
}
```

6. **Traverse Linked List:**

```
// start traversing from first node
trav = head;
do
{
      // visit current node
      printf("%d, ", trav->data);
      // go to the next node
      trav = trav->next;
} while(trav!=head);
// repeat until end of list is reached
```

- **Doubly Linear Linked List**

  Each node contains data, address of next node and address of previous node.

1. **Add First: O(1)**

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
      head = newnode;
else
{
      // newnode's next --> head
      newnode->next = head;
      // cur head's prev --> newnode
      head->prev = newnode;
```

```
        // head --> newnode
        head = newnode;
}
```

2. Add Last: O(n)

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
        head = newnode;
else
{
        // traverse till last node
        trav = head;
        while(trav->next!=NULL)
                trav = trav->next;
        // trav's next --> newnode
        trav->next = newnode;
        // newnode's prev --> last node (trav)
        newnode->prev = trav;
}
```

3. Insert at position: O(n)

```
// if list is empty or first pos, newnode is first node.
if(head==NULL || pos==1)
        addfirst(val);
else
{
        // create and initialize node
        newnode = new node(val);
        // traverse till pos-1 node
        trav = head;
        for(i=1; i<pos-1; i++)
                trav = trav->next;
        // newnode's next --> trav's next
        newnode->next = trav->next;
        // newnode's prev --> trav
        newnode->prev = trav;
        // next node's prev --> newnode
        if(trav->next!=NULL)
                trav->next->prev = newnode;
        // trav's next --> newnode
        trav->next = newnode;
}
```

4. **Delete First: O(1)**

```
if(head!=NULL)
{
      // keep address of first node in temp pointer
      temp = head;
      // take head to next node.
      head = head->next;
      // delete temp node.
      delete temp;
      // if node deleted is not last, make its prev NULL
      if(head!=NULL)
            head->prev = NULL;
}
```

5. **Delete At Position: O(n)**

```
// if node to be deleted is first or list is empty
if(pos==1 || head==NULL)
      delfirst();
else
{
      // traverse till pos
      trav = head;
      for(i=1; i<pos; i++)
            trav = trav->next;
      // trav's prev node's next --> trav's next
      trav->prev->next = trav->next;
      // trav's next node's prev --> trav's prev
      if(trav->next!=NULL)
            trav->next->prev = trav->prev;
      // delete trav node
      delete trav;
}
```

6. **Delete All: O(n)**

```
// delete all nodes one by one, until list become empty
while(head!=NULL)
      delfirst();
```

7. **Traverse Linked List:**

```
// start traversing from first node
trav = head;
while(trav!=NULL)
{
      // visit current node
```

```
        printf("%d, ", trav->data);
        // go to the next node
        trav = trav->next;
} // repeat until end of list is reached
```

8. **Traverse Linked List – In Reverse Direction:**

```
// traverse till last node
trav = head;
while(trav->next!=NULL)
        trav = trav->next;
// start traversing from last node
while(trav!=NULL)
{
        // visit current node
        printf("%d, ", trav->data);
        // go to the prev node
        trav = trav->prev;
} // repeat until beginning of list is reached
```

- **Doubly Circular Linked List**

  Each node contains data, address of next node and address of previous node. Last node's next contains address of first node, while first node's prev contains address of last node.

1. **Add First: O(1)**

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
{
        head = newnode;
        newnode->next = newnode;
        newnode->prev = newnode;
}
else
{
        // jump to the last node
        tail = head->prev;
        // newnode's next --> head
        newnode->next = head;
        // newnode's prev --> tail
        newnode->prev = tail;
        // tail's next --> newnode
        tail->next = newnode;
        // cur head's prev --> newnode
        head->prev = newnode;
```

```
        // head --> newnode
        head = newnode;
}
```

2. **Add Last: O(1)**

```
// create and initialize node
newnode = new node(val);
// if list is empty, newnode is first node.
if(head==NULL)
{
        head = newnode;
        newnode->next = newnode;
        newnode->prev = newnode;
}
else
{
        // jump to the last node
        tail = head->prev;
        // newnode's next --> head
        newnode->next = head;
        // newnode's prev --> tail
        newnode->prev = tail;
        // tail's next --> newnode
        tail->next = newnode;
        // cur head's prev --> newnode
        head->prev = newnode;
}
```

	Array	Linked List
.	Array cannot grow or shrink dynamically (realloc() is not efficient).	Linked list can grow/shrink dynamically.
.	Array has contiguous memory. Hence random access is possible.	Nodes are not in contiguous memory. Hence only sequential access is allowed.
.	Arrays do not have memory overheads.	Linked lists have memory overheads e.g. each node contains address of 'next' node.
.	Insert/Delete at middle position need to shift remaining elements. Hence will be slower.	Insert/Delete at middle position need to modify links (next/prev pointers). Hence will be faster.
.	To deal to fixed number of elements and frequent random access, arrays are better.	To deal to dynamic number of elements and frequent insertion/deletion operators, linked lists are better.

	Singly Linked List with Head	Singly Linked List with Head & Tail
.	Add last operation is slower i.e. O(n)	Add last operation is faster i.e. O(1)
.	No overload.	Extra overhead i.e. tail pointer.

3.	Simplified list manipulation operations.	List manipulation need to consider tail pointer wherever appropriate.

	Singly Linked List	Doubly Linked List
1.	Unidirectional access.	Bi-directional access.
2.	For search and delete operation two pointers are required (Need to maintain prev pointer) during traversing.	For search and delete operation only one pointer is required (No need to maintain prev pointer) during traversing.
3.	Simplified list manipulation operations.	List manipulation need to consider prev pointer of each node in all operations.

	Doubly Linked List	Doubly Circular Linked List
1.	Traversing till last node is slower (add last operation) i.e. O(n)	Traversing till last node is faster (add last operations) i.e O(1)

Trees

- **Terminologies:**
  - Tree is a non-linear data structure in which one specially designated node is called as "root" (i.e. starting point of the tree) and remaining elements can be partitioned into "m" disjoint subsets so that each subset is a tree itself.
  - Number of child nodes for any node is called as "degree of a node".
  - Maximum degree of a child in tree is called as "degree of the tree".
  - All the nodes in the path from the root to that node, are called "ancestors" of that node.
  - All the nodes accessible from the given node, are called as "descendants" of that node.
  - Child nodes of the same parent are called as "Siblings".
  - Terminal node of the tree is called as "leaf node". Leaf node does not have child nodes.
  - Level of node:
    - Indicates position of the node in tree hierarchy.
    - Level of child = Level of parent + 1
    - Level of root = 0
  - The maximum level of a node is "Height of tree".
  - Tree with zero nodes (i.e. empty tree) is called as "Null Tree". Height of Null tree is -1.
- **Types of Trees:**
  - **Binary Tree:**
    - Tree in which each node has maximum two child nodes, is called as "binary tree".
    - Binary tree has degree 2. Hence it is also called as 2-tree.
  - **Binary Search Tree:**

- The binary tree in which left child node is always smaller and right child node is always greater or equal to the (parent) node; is called as "Binary Search Tree".
- Searching is faster. Time complexity: O(h). Where "h" is height of the tree.

- **Complete Binary Tree:**
  - Binary tree in which all leaf nodes are at the same level.
- **Strictly Binary Tree:**
  - Binary tree in which each non-leaf node has exact two child nodes.
- **Full Binary Tree:**
  - Binary tree with its full capacity for the given height.
  - In other words, adding one more node will increase height of the tree.
  - It is always complete as well as strictly binary tree.
  - Number of elements = $2^{(h+1)} - 1$
- **Almost complete binary tree:**
  - The binary tree which follows two conditions:
    - All leaf nodes are at level h or h-1.
    - All leaf nodes at last level (h), are aligned to left as much as possible.
- **Threaded Binary Tree:**
  - These trees will allow very fast in-order traversal (ascending or descending).
  - Binary tree in which right null link of each node is replaced by address of its in-order successor; such tree is called as "right threaded binary tree".
  - Binary tree in which left null link of each node is replaced by address of its in-order predecessor; such tree is called as "left threaded binary tree".
  - If right as well as left null links of each node is replaced by its in-order successor and in-order predecessor respectively; then such tree is called as "in threaded binary tree".
- **Skewed Binary Tree:**
  - The binary tree in which only left link is used to keep address of child node (right link of each node is kept null), is called as "left skewed binary tree".
  - The binary tree in which only right link is used to keep address of child node (left link of each node is kept null), is called as "right skewed binary tree".
  - Height of skewed binary tree is maximum for given number of nodes. Hence searching will be too slow i.e. O(n)
- **Balanced binary tree:**
  - In binary search trees, searching is faster if height of the tree is kept as minimum as possible.
  - The binary search tree with minimum possible height (for given number of nodes) is called as "balanced binary search tree" or "height balanced binary search tree".
  - Balance factor of node = height of left sub tree - height of right sub tree.
  - The binary search tree, in which balance factor of each node is in range of -1 to 1, is called as "balanced binary search tree".

- On any binary search tree, series of appropriate left rotations and right rotations on appropriate nodes can convert it into "balanced binary search tree".
  - If balance factor of a node < -1, apply left rotation on that node.
  - If balance factor of a node > +1, apply right rotation on that node.

- **AVL Tree:**
  - Invented by "Greogy Adelson-Velsky" and "E.M. Landis".
  - Self-balancing binary search tree.
  - After each insert or delete operation, appropriate rotations are applied to make tree balanced again.

- **B Tree:**
  - The B-tree is a generalization of a binary search tree in that a node can have more than two children.
  - In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range.
  - In order to maintain the pre-defined range, internal nodes may be joined or split.
  - Each internal node of a B-tree will contain a number of keys. The keys act as separation values which divide its sub-trees.
  - A B-tree is kept balanced by requiring that all leaf nodes be at the same depth (level).

- **B+ Tree:**
  - Like B trees but with few changes as follows:
  - In the B+ tree, copies of the keys are stored in the internal nodes.
  - The keys and records are stored in leaves.
  - A leaf node may include a pointer to the next leaf node to speed sequential access.

Binary Search Tree:

- Implementation (Design) of binary search tree:

```
class tree;
class node
{
private:
   int data;
   node *left, *right;
public:
   node(int val=0);
   friend class tree;
};
class tree
{
private:
   node *root;
public:
   tree();
   void add(int val); // add node
   node* search(int key); // search node
   node* search(int key, node **parent); // search node with parent
   void preorder(node *trav); // preorder traversal
   void preorder(); // wrapper for preorder(root);
   void inorder(node *trav); // inorder traversal
   void inorder(); // wrapper for inorder(root);
   void postorder(node *trav); // postorder traversal
   void postorder();     // wrapper for postorder(root);
   void height(node *trav); // height of tree
   void height();   // wrapper for height(root);
   void clear(node *trav); // delete node and its descendants
   void clear();    // wrapper for clear(root);
   void preorder_nonrec(); // preorder traversal non-recursive
   void inorder_nonrec(); // inorder traversal non-recursive
   void postorder_nonrec(); // postorder traversal non-recursive
   void del(int key); // search and delete node
};
```

- Add a node in tree:

```
void add(int val)
{
        // create and initialize node
        node *newnode = new node(val);
        // if tree is empty
        if (root == NULL)
```

```
                root = newnode;
        else
        {
                node *trav = root;
                while (true)
                {
                        if (val < trav->data)
                        {
                                if (trav->left == NULL)
                                {
                                        trav->left = newnode;
                                        break;
                                }
                                trav = trav->left;
                        }
                        else
                        {
                                if (trav->right == NULL)
                                {
                                        trav->right = newnode;
                                        break;
                                }
                                trav = trav->right;
                        }
                }
        }
}
```

- **Tree Traversal:**
  - Pre-order: Parent , Left, Right
  - In-order: Left, Parent, Right
  - Post-order: Left, Right, Parent

```
void preorder(node *trav)
{
        if (trav == NULL)
                return;
        printf("%d, ", trav->data);
        preorder(trav->left);
        preorder(trav->right);
}
void inorder(node *trav)
{
        if (trav == NULL)
```

```
            return;
        inorder(trav->left);
        printf("%d, ", trav->data);
        inorder(trav->right);
}
void postorder(node *trav)
{
        if (trav == NULL)
            return;
        postorder(trav->left);
        postorder(trav->right);
        printf("%d, ", trav->data);
}
```

- **Delete All Nodes in Tree:**

```
void clear(node *trav)
{
        if (trav == NULL)
            return;
        clear(trav->left);
        clear(trav->right);
        delete trav;
}
void clear()
{
        clear(root);
        root = NULL;
}
```

- **Height of tree:**

```
int height(node *trav)
{
        if (trav == NULL)
            return -1;
        int hl = height(trav->left);
        int hr = height(trav->right);
        int max = (hl > hr ? hl : hr);
        return max + 1;
}
int height()
{
        return height(root);
}
```

- **Search a node with its parent:**

```
node* search(int key, node **parent)
{
	node *trav = root;
	*parent = NULL;
	while (trav != NULL)
	{
		if (key == trav->data)
			return trav;
		*parent = trav;
		if (key < trav->data)
			trav = trav->left;
		else
			trav = trav->right;
	}
	*parent = NULL;
	return NULL;
}
```

- **Delete a node:**
  - if node's left is null:

```
if(temp->left==NULL)
{
	if(temp==root)
		root = temp->right;
	else if(temp==parent->left)
		parent->left = temp->right;
	else
		parent->right = temp->right;
}
```

  - if node's right is null:

```
if(temp->right==NULL)
{
	if(temp==root)
		root = temp->left;
	else if(temp==parent->left)
		parent->left = temp->left;
	else
		parent->right = temp->left;
}
```

  - if node's left as well as right is NOT null:

```
if(temp->left!=NULL && temp->right!=NULL)
{
```

```
        // find the succ of temp along with its (succ) parent
        parent = temp;
        succ = temp->right;
        while(succ->left!=NULL)
        {
                parent = succ;
                succ = succ->left;
        }
        // replace temp data by succ data
                temp->data = succ->data;
        // mark succ for deletion
        temp = succ;
}
```

- deleting node (full) - steps:
  - Search node to be deleted along with its parent.
  - If node's left as well as right is NOT null, deletion code for that.
  - If node's left is null, deletion code for that.
  - Else if node's right is null, deletion code for that.
  - delete memory of temp node.

- **Pre-order non-recursive:**

```
void preorder_nonrec()
{
        printf("NONREC PRE : ");
        node *trav = root;
        stack<node*> s;
        while (trav != NULL || !s.empty())
        {
                while (trav != NULL)
                {
                        printf("%d, ", trav->data);
                        if (trav->right != NULL)
                                s.push(trav->right);
                        trav = trav->left;
                }
                if (!s.empty())
                {
                        trav = s.top();
                        s.pop();
                }
        }
        printf("\n");
}
```

- **In-order non-recursive:**

```
void inorder_nonrec()
{
	printf("NONREC IN  : ");
	node *trav = root;
	stack<node*> s;
	while (trav != NULL || !s.empty())
	{
		while (trav != NULL)
		{
			s.push(trav);
			trav = trav->left;
		}
		if (!s.empty())
		{
			trav = s.top();   s.pop();
			printf("%d, ", trav->data);
			trav = trav->right;
		}
	}
	printf("\n");
}
```

- **Post-order non-recursive:**

```
void postorder_nonrec()
{
	printf("NONREC POST: ");
	node *trav = root;
	stack<node*> s;
	while (trav != NULL || !s.empty())
	{
		while (trav != NULL)
		{
			s.push(trav);
			trav = trav->left;
		}
		if (!s.empty())
		{
			trav = s.top(); s.pop();
			if (trav->right == NULL || trav->right->visited == true)
			{
				printf("%d, ", trav->data);
				trav->visited = true;
				trav = NULL;
			}
			else
			{
```

```
                        s.push(trav);
                        trav = trav->right;
                    }
                }
            }
            printf("\n");
}
```

- Depth First Search:

```
node* bfs(int key)
{
        node *trav;
        queue<node*> q;
        //push root on queue
        q.push(root);
        //repeat until queue is empty
        while (!q.empty())
        {
                // pop an ele from queue
                trav = q.front();      q.pop();
                // compare with ele to search
                if (key == trav->data)
                        return trav;
                // if trav has left child, push on queue
                if (trav->left != NULL)
                        q.push(trav->left);
                // if trav has right child, push on queue
                if (trav->right != NULL)
                        q.push(trav->right);
        }
        return NULL;
}
```

- Breadth First Search:

```
node* dfs(int key)
{
        node *trav;
        stack<node*> s;
        //push root on stack
        s.push(root);
        //repeat until stack is empty
        while (!s.empty())
        {
                // pop an ele from stack
                trav = s.top();  s.pop();
```

```
            // compare with ele to search
            if (key == trav->data)
                return trav;
            // if trav has right child, push on stack
            if (trav->right != NULL)
                s.push(trav->right);
            // if trav has left child, push on stack
            if (trav->left != NULL)
                s.push(trav->left);
        }
        return NULL;
}
```

Array Implementation of tree:

- A binary tree can be simulated in an array i.e. parent-child relationship can be maintained.
- Typically array begin with index 1. Then
  - parent index = child index / 2
  - left child index = parent index * 2
  - right child index = parent index * 2 + 1
- Array implementation of almost complete binary tree, is called as "Heap" data structure.
- If each parent element in heap is greater than both of its child elements, then it is called as "Max Heap".
- If each parent element in heap is less than both of its child elements, then it is called as "Min Heap".
- Heap sort algorithm uses Max heap (ascending sort) or Min heap (descending sort).

Hash-table

- Hash-table is data structure used for fastest searching mechanism.
- In hash-table, data is stored in key value pair, so that for a given key value can be searched in fastest possible time.
- Ideal time complexity of searching/inserting in hash-table = O(1).
- Since data is stored as key-value pair, it is an "associative" data structure.
- Typically hash table is implemented as array and element is stored at appropriate index (calculated using hash function).
- Hash function:
  - The mathematical function of "key", which decides the slot (index) of the table (array) to be used for storing key-value pair.
  - e.g. Key is "roll" and value is "Student" object; simplest hash function can be $h(r) = r - 1$
  - In this case, Student with roll number 51 will be stored at index 50.
- Ideal hash function gives different slot for different keys.

- However, many times different key values can compete for the same slot of the table. This situation is called as "collision".
- e.g. Key is "roll" and value is "Student" object and table has 10 slots. So hash function can be h(r) = r % 10
- In this case, student with roll 52, 32, 72 will result in the same slot number i.e. slot 2.
- Load Factor = Number of Entries / Number of slots
- Collision can be handled by one of the following mechanisms:
  - Open Addressing:
    - If slot computed by hash function is already in use; then another mathematical function can be used for next possible empty slot. This function is called as "rehashing" function.
    - In above example: h(r) = r % 10
    - Then, Rehashing function can be : f(i) = (i + 1) % 10 i.e. Immediately next slot.
    - The rehashing function can be a linear function (as example). In that case, open addressing is also called as "linear probing".
    - If required, rehashing function can be a quadratic or any other polynomial function.
    - This mechanism can be used if and only if, number of entries is less than number of slots in hash-table. In other words, open addressing can be used only if load factor <= 1.
  - Chaining:
    - Each slot in the hash table maintains a linked list of key-value pairs, which will compete for the same slot. This linked list is called as "buckets".
    - This mechanism can be used irrespective of load factor.
    - If number of collisions is more, time required for searching/insertion will increase. In other words, time complexity will be deviated from O(1).

Graph

- **Terminologies:**
  - Graph is a non-linear data structure having set of vertices (nodes) and set of edges (arcs).
    - G={V, E} Where V is set of vertices and E is set of edges.
  - Vertex (node) is an element in graph.
    - V = {A, B, C, D, E, F}
  - Edge (arc) is a line connecting two vertices (nodes).
    - E = {(A,B), (A,C), (A,D), (B,C), (B,E), (D,E), (D,F)}
  - Vertex A is said to be adjacent to B, if and only if there is an edge from A to B.
  - Number of vertices adjacent to given node, is called as "degree of node".
  - Set of edges connecting any two vertices is called as "Path" between those two vertices.
    - Path between A to D = { (A,C), (A,B), (B,E), (E,D) }
  - Set of edges connecting a node to itself is called as "Cycle".
    - {(A,B), (B,E), (E,D), (D,A)}
  - An edge connecting a node to itself is called as "Loop". Loop is smallest cycle.

- **Types of Graph:**
  - **Simple graph:**
    - If graph do not have multiple edges between adjacent nodes and no loops present, then it is called as "Simple graph".
  - **Complete graph:**

Prasanna Wadekar

- A simple graph in which each node is adjacent with every other node, then it is called as "Complete Graph".
- Number of edges = n (n-1) / 2

- **Connected graph:**
  - A simple graph in which there is some path exists between any two vertices, is called as "Connected graph".
- **Multi-graph:**
  - If in a graph, there are multiple edges between two adjacent nodes or loops, then it is called as "multi-graph" (See above diagram).

Prasanna

- **Weighted graph:**
  - A graph in which each edge is associated with a number (i.e. weight) is called as "weighted graph".
  - Examples:
    - If two vertices represents cities, then weight will represent distance between those two cities.
    - If two vertices represents connecting points in an electronic circuit, then weight will represent resistance between those two points.
  - The sum of weights of all edges in a path is called as "weight of the path".
  - Dijkstra's algorithm is used to find shortest path in two vertices.

- **Directed Graph:**
  - A graph in which each edge has some edge is called as "Directed Graph".
  - In below example:
    - E = {<A,B>, <B,C>, <C,E>, <D,B>, <E,D>, <E,F>}
  - Note that there is an edge from A to B; so A is adjacent to B. In this case B is not adjacent to A.
  - For node E,
    - Number of edges originated from E = 2 called as "out-degree".
    - Number of edges terminated on E = 1 called as "in-degree".
  - Directed graphs are also called as "di-graphs".

- **Directed Weighted Graph:**

- The graph in which each edge has some weight as well as direction, it is called as "directed weighted graph".
- It is called as "Network".

- **Spanning Tree:**
  - A tree can be defined as a graph without any cycle.
  - Any connected graph can be converted into a tree by removing few edges.
  - A tree having all vertices of the graph but eliminated one or more edges is called as "Spanning tree" of the graph.
  - One graph can have multiple spanning trees (by removing different edges).
  - Sum of weights of all edges in spanning tree is weight of the spanning tree.
  - Algorithms used to calculate minimum weight spanning tree are:
    - Prim's algorithm
    - Kruskal's algorithm
- **Forest:**
  - A forest is an undirected graph, all of whose connected components are trees.
  - In other words, the graph consists of a disjoint union of trees. Equivalently, a forest is an undirected cycle-free graph.

Graph Presentation/Creation

1. **Adjacency List [Linked list]**
   a. In this method, array is used to represent vertices. Also each vertex holds a linked list for all vertices that are adjacent to that vertex.

   b. Total space required is V + E.

2. **Adjacency Matrix [2-D array]**
   a. A graph with "n" vertices can be represented by n x n matrix.

b. If there is an edge between two vertices '1' & '2', then it is represented by 1 in the matrix i.e. mat[1][2] = 1. If there is no edge, corresponding element in matrix will be zero.

c. For undirected graphs, matrix will be symmetric across the diagonal.

d. For above graph matrix will be:

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	0	1	0
C	1	1	0	0	0	0
D	1	0	0	0	1	1
E	0	1	0	1	0	0
F	0	0	0	1	0	0

e. In case of weighted graph, matrix element corresponding to an edge stores weight (number). If there is no edge connected, matrix element will be ∞.

f. Total space required is N^2.