* System.out.println
Class name in java.lang
static variable of type PrintStream in the System

Date:
Page No.:

Introduction to Java Programming

* originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995.

Object oriented Programming

↳ i) Encapsulation
ii) Abstraction and Implementation Hiding
iii) Inheritence, Dynamic binding and polymorphism
iv) Overriding and overloading

* Java — Programming language

Platform
{ JRE → Java Runtime Environment

Features of Java

i) object oriented
OOP → object, class, Inheritence, Polymorphism, Encapsulation

ii) Platform independent
Platform to run application
↓ Runtime
→ Application Programming Interface (API)

Java —compile→ byte code → Machine independent
Windows
MAC { Java virtual machine }
OS

class
↓
public — access modifier — JVM can access it outside class
static — called without any object — starting
void
main (String args[])
Command line args

iii) Simple
Java does not have pointers

iv) Secure
⇒ Java does not have pointers so you can't access your memory directly
⇒ If there is any problem in the Java program then the problem will remain in the Java virtual machine only and it will not effect the system

v) Architectural Neutral
Because there is no implementation of dependent features that is the eg. of size of primitive type is set.
ie. no need of different type of sizes of diff. data types

vi) Portable

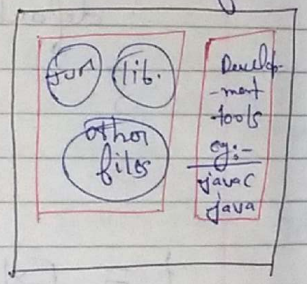
vii) Robust
(no memory management)
Java comes with automatic garbage collection and strong exception handling
(to correct errors)

viii) Multi-threaded
Programme { Programme can be divided into threads }
t1 threads t2
we can run threads concurrently
→ leads to optimized programming

JAVA SE (edition) (2)

Difference b/w JDK, JRE and JVM

JVM	JRE	JDK
- Java Virtual machine provides runtime environment in which Java Byte code execute.	- Java runtime environment	- Java development kit
- Abstract machine (it is available for both software and hardware)	- Implementation of JVM	- Physically present
- Provides runtime environment in which java bytecode can be executed	- physically present	- Contains JRE + development tools → superset of JRE
- Platform dependent i.e. can be run in any machine	- contains libraries + other files that JVM uses	



JRE

JDK

Functions:
1. Loads codes
2. Verifies code
3. Executes codes
4. Provides runtime environment

* It doesn't understand source code, there .java → .class (Byte code)

It allows to run applets and application

doesn't contain compilers or debuggers to develop apps.

→ JDK's provided by oracle and other organizations also

Compilation and Execution Process of Java program

Compiletime

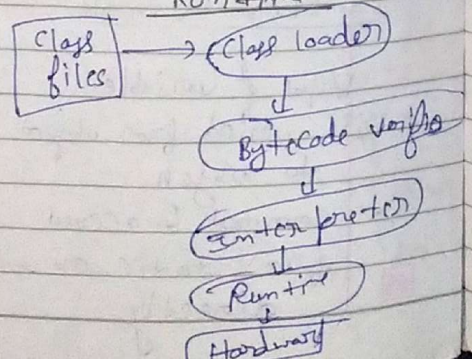
java code → Byte code

java compiler

```
class abc
{
  public static void main()
  {
  }
}
```

abc.java → Compilation → Byte code → abc.class

Runtime

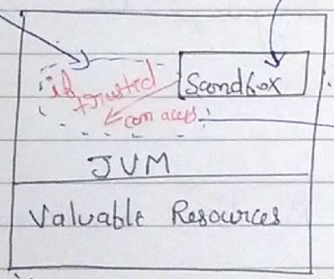


* For command prompt

javac filename.java → filename.class → bytecode

Java Sandbox Model [Explain & why it is important]

- Security model of Java
- Provided restricted environment
- Local code is trusted to have full access to vital system resources (such as file system), it do not hav-
- Download remote code (Applet) is not trusted and can access only limited resources provided inside the sandbox.



→ Modified version of Sandbox

It relies on three tired def. If any of fail Security model completely compromises and vulnerable to attack

→ Byte code Verifier
→ Applet class loader
→ Security manager

Not in main syllabus

How to pass Command Line Arguments in JAVA (used in command prompt)

```
Class ABC
{
  public static void main (String args[])
  {
    System.out.println ("Argument " + args[0]);
  }
}
```

o/p
Array out of Bounds Exception — why?

due to it is expecting atleast one argument to be passed during run time

∴ Always command line arguments of string type

> javac ABC.java → ABC.class
> java ABC Hello
> now o/p
Argument Hello

For using integer

```
int i, j, Sum = 0;
i = Integer.parseInt(args[0])
j = Integer.parseInt(args[1]);
System.out.println("Sum of i and j is: "+Sum);
```

Not in syllabus

How to take Input from User in Java

Follow given steps :-

1. Import java.util.Scanner
2. Create object of Scanner class
3. Use in-built methods to take i/p
   - ↳ nextInt();
   - ↳ next();

Example

```
import java.util.Scanner;
class input1
{
  p.s.v.m (String args[])
  {
    int x;
    String S;
  Scanner SC1 = new Scanner(System.in);
    x = SC1.nextInt()
    S = SC1.next();
    S.o.p(x);
    S.o.p(S);
  }
}
```

Decision Making th. Branching

if statement	if else	if else if Ladder	Nested if statement
`if (cond-)` `{ stat }`	`if (cond-) { stat.. } else { stat. }`	`if (comd-) { stat.. } else if (Cond-) { stat. } else { stat. }`	`if (cond- { if(Cond) { stat. } } }`

Variable Rules :- 1. must not begin with a digit.
2. Uppercase & lowercase are distinct.
3. It should not be a keyword.
4. White space is not allowed
5. It can be of any length.

Scope
Declaration
Initialization
Value assignment
Call

Switch Statement

```
Switch (Expression)
Case value 1: stat 1;
      |
Case value 2: Stat 2;
      |
Case value 3: Stat 3;
      |
  default statement
      ;
}
```

? : Statement :- General form is

i = Expression 1 ? Exp2 : Exp3

(True / False)

→ A variable is an identifier that denotes a storage location used to store a data value.

→ It can the diff. values during the execution of program.

Types of Variables in Java

- Primitive variables → used to ref. primitive values eg. int x = 100;
- Reference Variables → used to refer objects.

Variables

ABC a1 = new ABC(); (ABC: class, obj)

- Instance Variable
- Static Variable (on basis of declaration / pos.)
- Local Variable

Instance Variable	Static Variable	Local Variable
- Value of variable is varied from object to object ← cannot be accessed from static area directly	- Value is not varied from object to object. - Known as class-level variables	- Created inside a block or constructs, methods. - Initialization is necessary before using it.

↳ value of variable is varied object to object.

Instance Variable

↳ Scope of object
↳ should be declared within the class directly, but outside any block, method or constructor
→ cannot be access from static method directly

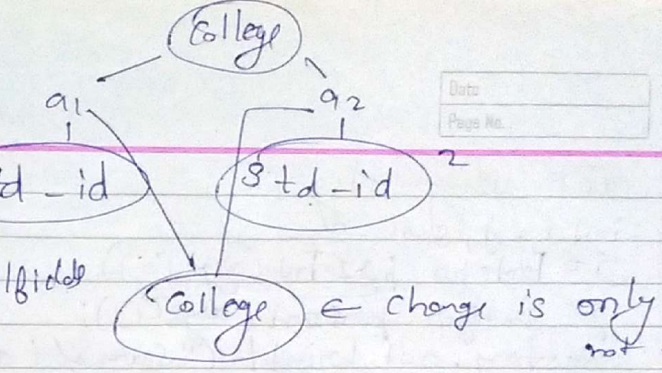
```
Class ABC
{
  int i=100;
  p.s.v.m (String a[]) ] static method
  {
    ABC a1 = new ABC();
    S.O.P(i) → compile time error
    SOP(a1.i);
  }
}
```

a1.i = 200
i
a2.i = 400

→ Initialization is not mandatory
int i; → i=0;

Static Variable

```
Static College
Class Abc
{ Static string college = "ASET"; ] static
  int std-id; ] instance
  p.s.v.m (String args[])
  {
    Abc a1 = new Abc();
    Abc a2 = new Abc();
    a1.std_id = 1;
    a2.std_id = 2;
  }
```



Class-level variable ∈ change is only here not for every

→ Scope of static variable = scope of class
→ value is not varied from object to object.
→ Declared inside class outside constructor or method
→ Initialization is not mandatory, jvm will initialize it

Local Variables → Scope [Block, method, Constructor]

* Initialization is mandatory

```
eg. Class ABC
{
  p.s.v.m (String [] a)
  {
    int i=10;
    void m1 ()
    {
    }
  }
}
```

X limited Scope & initialization mandatory

Wrapper Classes in Java

* It is used to convert primitive data types into object types and vice-versa in java.lang package.

Autoboxing: converting primitive data type to object type

Unboxing: converting object type to primitive data type

Eight (8) classes of java.lang package are known as wrapper class in Java.

Primitive type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Example

Primitive to wrapper

```
class Wrapper1 {
  p.s.v.m (String[] a) {
    int i = 100;
Integer j = Integer.valueOf(i);   // primitive
Integer ab = i; // Autoboxing
S.o.p();
  }
}
```

Wrapper to primitive

```
class Wrapper2 {
  p.s.v.m (String[] a) {
    Integer i = new Integer(b);
int ab = i.intValue();
int j = i; // Unboxing
S.o.p();
  }
}
```

Loop Control / Iteration Statements

* A loop statement allows us to execute a statement or a group of statements multiple times.

Types of Loops

- Entry-Controlled loop (cond^n is checked at entry time)
  - while loop
  - for loop
- Exit-Control loop (cond^n is checked during exit)
  - do-while loop

```
while loop
while(condn)
{
  ——
}

for loop
for(int i; condn, increment)
{
  ——
}

do-while loop
do
{
  ——
} while(condn)
```

→ if a stat. has to execute atleast once

Break Statement in Java

Whenever a break statement is encountered, all the remaining statements in the loop following the break statement are skipped and control passes to the first statement outside the loop.

Example

```
class ABC {
  p.s.v.m (String args[]) {
    for (int i=0; i<=3; i++) {
      if (i==2) break;
      sop(i);
    }
    s.o.p("Exit from Break");
  }
}
```

o/p

```
0
1
Exit from Break
```

Break statement in Nested Loops

In the case of nested loops, if break statement is used, control will be transferred only out of innermost loop

Example

```
class ABC {
    p.s.v.m (String a[])
    {for (int i=0; i<3; i++) {
        for (int j=0; j<2; j++) {
            ---- break;
        }
    }
}
```

Labelled Break Statements :-

```
Class breaktest {
    p.s.v.m (String[] a) {
    outer: for (int i=0; i<3; i++) {
        inner: for (int j=0; j<2; j++) {
            }
        }
    }
```

Break from both loops → break outer;

break inner;) Break from inner step

Continue Statement :

It is used to bypass the remainder of the code for some cond^n for a particular loop from a certain point but continue the execution of the loop

Example

```
class Abc {
    p.s.v.m (String a[]) {
        for (int i=0; i<=3; i++) {
            if (i==2) continue;
            S.o.P (i);
        }
    }
}
```

o/p	o/p
0	0
1	1
3	

For labelled continue see the labelled break

Classes and objects in Java

→ A class is a structure from which objects can be initialized.
→ properties of class = attributes → variables
→ Functions which act on attributes = methods / functions

Declaring a class

keyword

```
Class Faculty {
    int fac_id;
    String fac_name;
}
```

Creating objects

→ An object is an instance of a class
→ new operator is used

```
Faculty fac1 = new Faculty();
Faculty fac2 = new Faculty();
```

fac1.id=2
fac1.fac_name = 'Abc'
fac2.id=3
fac2.fac_name = 'xyz'

Adding Method to class

General form is

```
void/int/ method name (parameter list)
{
    method body
}
```

Calling → fac1.m1();
method name

Java class Loader Architecture

Page No. :
Date :

Java class Loader loads a java class file into java virtual Machine. It is also responsible for causing the exception of Class Notfound Exception.

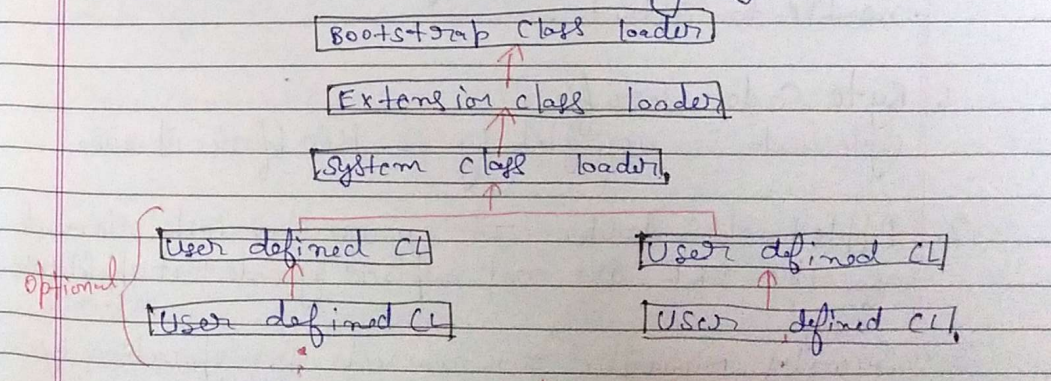
Types of (Hierarchy) Java class Loaders

Java class loaders can be broadly classified into below categories:

- **Bootstrap Class Loader**: Bootstrap CL loads java core classes like java.lang, java.util etc. These are classes that are part of JRE. Bootstrap CL is native implementation and so they may differ across different JVM

- **Extensions class Loader**: JAVA_HOME /jre/lib/ext contains jar packages that are extensions of standard core java classes. Extensions class loader loads classes from this ext folder. Using system environment properly java.ext.dirs you can add 'ext' folder and jar files to be loaded using extensions class loader.

- **System Class Loader**: Java classes that are available in Java class path loaded using system CL.



Class loader architecture

Page No.:
Date:

Java Sandbox Model

* A security measure in the JRE. The Sandbox is a set of rules that are used when creating an applet that prevents certain functions when the applet is sent as part of a web page.

* If the applet is allowed unlimited access to memory & operating system resources, it can do harm in the hands of someone with malicious intend.

* Sandbox creates an environment in which there are strict limitations on what system resources the applet can request or access. Sandboxes are used when executable code comes from unknown or untrusted resources and allow the user to run untrusted code safely.

Java Sandbox - relies of three-tired defence. If any one of these three elements fails, the Security model is completly compromised and vulnerable to attack.

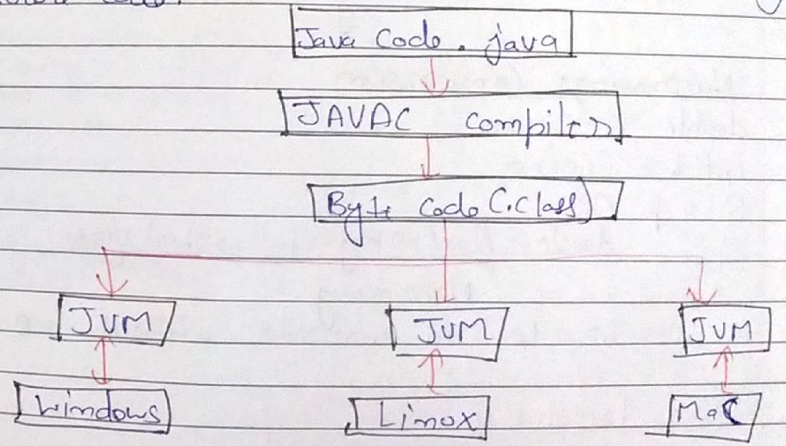
1. Byte Code Verifier
   Byte code is verified by Sandbox before it runs.
2. Applet class loader - It ensures that imp elements of the JRE are not replaced by code that applet tries to install.
3. Security Manager - It can veto the operation by generating a security exception.

Page No.:
Date:

Java Bytecode

* Bytecode is nothing but the intermediate representation of Java source code which is produced by Java compiler by compiling that source code.

* Byte code is platform independent which is stored in .class file which is created after compiling the source code.



Type Conversion

Changing a value from one data type to another data type is known as data type conversion.

Type conversions are either widening or narrowing, depend upon the data capacities of the data types involved.

It can be done by two ways implicitly or explicitly.

Widening Conversion
If a value of narrower (lower size) data type

converted to a value of a broader (higher size) data type without loss of info. This is implicit casting done by JVM

eg. int a = 100
double b = a;
System.out.println(b);

byte → short → int → long → float → double
widening

Narrowing Conversion
double b = 100.7;
int b = (int) a
S.o.p(b)

double → float → long → int → short → byte
Narrowing

broader to narrow without loss of data

Java Primitive Data Types with Size

Type	Size in Bytes	Range
Byte	1 byte	-128 to 127
Short	2 bytes	-32,768 to 32,768
Int	4 bytes	-2,147,483,648 to +
Long	8 bytes	-9,223,372,036,854,775,808 to +
Float	4 bytes	±3.4028234 7E+38F
Double	8 bytes	± 1.7976931348623157 E+308
Char	2 byte	0 to 65,536 (unsigned)

Access Modifiers in Java

It specifies accessibility (scope) of a data member, method, constructor or class.

4 types of Java access modifiers

1. private access modifier
2. Default access modifier
3. Protected access modifier
4. Public access modifier

1. If you make any class constructor private, you cannot create the instance of that class from outside the class

```
Class A {
private A() {}
// rest same
}
```

2. If you do not use modifier it is treated as default

⇒ package pack

```
Class A {
void msg()
{ S.o.p("Hello"); }
}

package mypack
import pack.*
Class B {
psvm(String args[]){
A obj = new A(); // CT
obj.msg(); // CT
}
}
```

Same // error

It is accessible only within package

3. It can be access on data mem, method, const., not on class

```
Class B extends A{
B obj = new B();
```

o/p Hello

4. Public access modifier accessible everywhere. It has widest scope among all

Access modifiers	within class	within pkg	outside pkg by Subclass	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

This keyword

In Java, this keyword is a refrence variable that refers to the current object.

Features of this keyword :-

1. this can be used to refer current class instance variable
2. this() can be used to invoke current class method. (implicitly)
3. this() can be used to invoke current class constructor.
4. this() can be passed as an argument in the method call.
5. this() can be passed as an argument in the constructor call
6. this can be used to return the current class instance from the method.

eg.

```
Class A {
  int rollno.;
  A (int rollno.) {
    this.rollno = rollno;
  }
```

Super Keyword

The super keyword in java is a reference variable which is used to refer immidiate parent class object.

Features of Super Keyword

1. Super can be used to refer immidiate parent class instance variable.
2. Super() can be used to invoke immidiate parent class method.
3. Super() can be used to invoke immidiate parent class constructor.
4. It cannot be passed as an arg. in the method call.
5. It cannot be passed as an argument in the constr. call.
6. It cannot ~~return~~ used to return any value.

eg.

```
Class Animal {
String color = "white";
}
Class Dog extends Animal {
String color = "black";
void printColor() {
S.o.p (color)
S.o.p(super.color) }}
Class test {
psvm (S args[]) {
Dog d = new Dog();
d.printcolor()
}}
```

o/p : black
white

Page No.:
Date:

Constructor

* Constructor in java is a special type of method that is used to initialize the object.

* It is invoked at the time of object creation. It constructs the value i.e. provides data for the object

Rules for creating Java Constructor

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of Java constructors

1. Default constructor (no-arg constructor)
   A constructor having no parameter is known as default constructor.

   If there is no constructor in the class, compiler automatically creates a default constructor

   It provides the default values to the object like 0, null etc. depending on the type.

   Syntax - <class_name>(){?

2. Parameterized constructor
   A constructor that have parameter

   It is used to provide different values to distinct objects

Page No.:
Date:

Example

```
Class A{
    int id;
    String name;
    A(int i, String n){
        id = i;
        name = n;
    }
}
```

3. Copy Constructor
   There is no copy constructor in java, other ways to copy the values of one object in another in java. are:

   - By constructor
   - By assigning the values of one object into another.
   - By clone() method of object class

Example

```
class A{
    int id;
    String name;
    A(int i, String n){
        id = i;
        name = n;
    }
    A(A a){
        id = a.id;
        name = a.name;
    }
}
```

Page No.:
Date:

Constructor overloading

It is a technique in Java in which a class can have any no. constructors that differ in parameter lists. The compiler differentiate these ~~no of p~~ constructors by taking into account the no. of parameters in the list and their type.

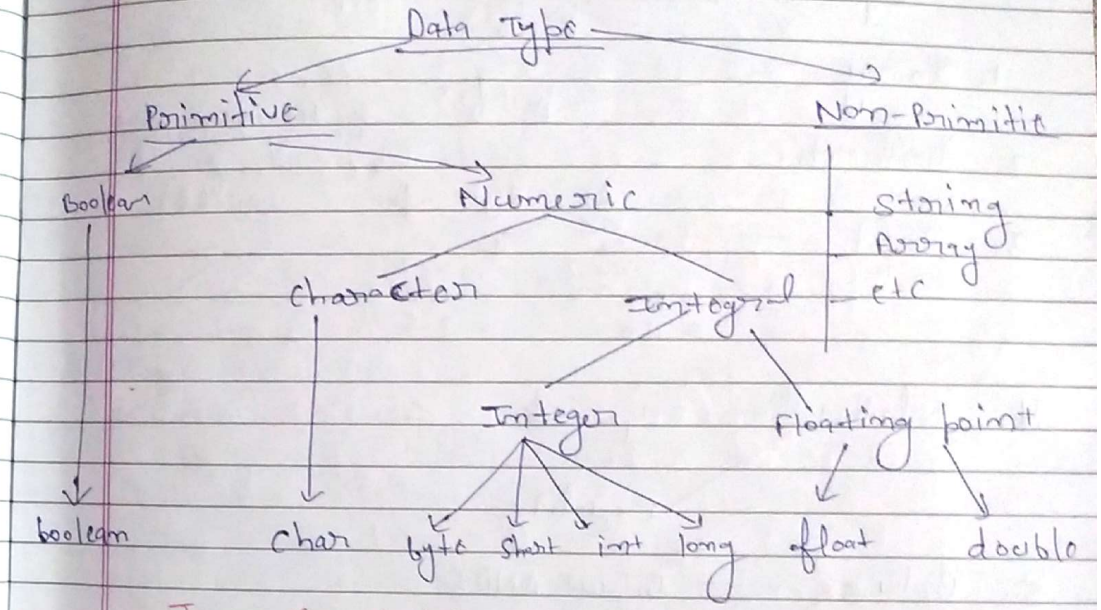
Example

```
Class A {
    int id;
    int age;
    String name;

    A (int i, String n) {
        id = i;
        name = n;
    }

    A (int i, String n, int a) {
        id = i;
        name = n;
        age = a;
    }
}
```

Java Constructor	Java Method
1. Initialize state of an object	1. It is used to expose behaviour of an object
2. It must not have return type.	2. It must have return type.
3. Invoked implicitly	3. Invoked explicitly
4. It's name must be same as that of class	4. It may or may not have same name as that of class
5. Compiler provides a default constructor if you don't have any constructor.	5. It is not provided by compiler in any case

Page No.:
Date:

Datatypes Classification



Java Annotations

Java Annotation is a tag that represents the metadata i.e attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Built in Java Annotations used in java code @Override & assures that subclass is overriding the parent class method, if it is not so compile time errors occure), @SupressWarnings, @Deprecated

Built in Java Annotations used in other annotations @Target, @Retention, @Inherited, @Documented

Operators

1. Unary — postfix — exp++/--
   prefix — ++/-- exp
   ~ decrement value by arithmetic rules
   ! (boolean of variable)
2. Arithmetic
3. Shift — Left shift <<, Right shift >>, >>>
   10<<2 = 10*2^2 = 40
   10>>2 = 10/2^2 = 2.5
   20>>>2 = +5
   -20>>>2 = -1073741818
4. Relational — comparison < > <= >=
   Equality == !=
5. Bitwise — Bitwise AND &
   Bitwise exclusive OR ^
   Bitwise inclusive OR |
   Note: In true condition both the condition are checked
6. Logical — logical AND &&
   logical OR ||
   first condition is true, second is not checked
7. Ternary — ? :
   a=10, b=5
   int min = (a<b)?a:b;
   o/p 5
8. Assignment =
   += -= *= /= => a=a+4; a+=4
   %= , &= , ^=
   |= <<= >>=
   >>>=

Array

[Index based, first element is zero]

Array is a collection of similar type of elements that have contiguous memory location.

Java array is an object that contains object of similar data type. It is a DS which store element

Advantage - 1. It can make the code optimized, we can retrieve or sort data easily.
2. We can get any data location at any index position

Disadvantage - We can store only fixed size of elements in the array. It doesn't grow its size at runtime

1D Array

int a[] = {33, 3, 4, 5};

2D Array

int arr[][] = {1,2,3,2,5,4,6};

eg. of Sorting, for loop value taking & print

matrix addition

→ // creating matrix (array), another matrix (array) to store in c = new int[][]

→ add c[i][j] = a[i][j] + b[i][j]

Java Packages

A Java package is a group of similar type of classes, interface & sub packages. Package in Java can be categorized in two forms:

1. built in package
2. User-defined package

The package keyword is used to create a package in Java.

```
package mypack;
public class simple{
p.s.v.m(String args[]){
    s.o.p("welcome to package");
    }
}
```

package → import → class

Advantage of Java package

1. To categorize classes & Interface so that they can be easily maintained.
2. provides access protection and removes naming collision

Java packages with their classes

1. Lang: Number, object
2. Awt: TextField, TextArea
3. Net: InetAddress, Socket
4. IO: DataInputStream and BufferedReader
5. Util: Date, Scanner

Utility Package in Java

java.util package defines a no. of useful classes, primarily collection classes that are useful for working with groups of objects. The package should not be considered merely a utility package that is separate from the rest of the language; In fact Java depends directly on several of the classes in this package

Methods declared in Collection Interface	Description
1. Public boolean add (Object element)	- to insert element in collection
2. Public boolean addAll (Collection c)	- to insert the specified collective elements in invoking collection
3. Public boolean remove (Object element)	- to delete an ele from collection
4. Public boolean removeAll (Collection c)	- to delete all the elements of specified collection from invoking collection
5. Public Iterator iterator	- returns an Iterator

The final Java package, java.util, contains a collection of utility classes

1. DS classes: variety of useful classes implementing standard Computer Science DS: including BitSet, Dict., Hashtable, Stack & Vector.
2. Date: Use of Date class to create & manipulate calendar data in a system independent fashion.
3. String tokenizer: Converts a string to text into token
4. Properties: Used by System class to implement System properties
5. Observer & Observable: classes that implements the observer interface can "watch" observable objects for state changes. When observable object changes, it notifies all of its observers of the change.
6. Random number generator: Random class implements a Random number generator.
7. Enumeration: Enumeration interface defines a generic programming interface for iterating through a set of values

Page No. :
Date :

Difference between JAVA & C++

Java	C++
1. Java does not support pointers, templates, unions, operators, operator overloading, structures etc.	1. C++ supports structures, unions, templates, operator overloading, pointers, operator etc.
2. It supports automatic garbage collection. It does not support destructors as C++ does.	2. It supports destructor, which is automatically invoked when the scope of an object is over.
3. It does not support conditional compilation and inclusion	3. Conditional inclusion (#ifdef #ifndef type) is one of the main features of C++.
4. It has built in support for threads, there is a thread class that you inherit to create a new thread and override the run() method.	4. It has no built in support for threads. C++ relies on non-standard third party libraries for thread support.
5. Java doesn't provide multiple inheritance, it can be done by using interfaces.	5. It does support multiple inheritance
6. It has method overloading but no operator overloading	6. It supports both method overloading and operator overloading.

Page No. :
Date :

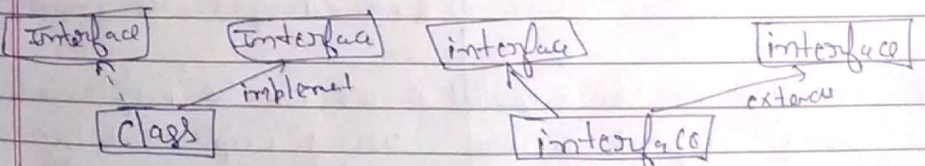
Interfaces [It has static constants and abstract methods]

multiple inheritance

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body

Three reasons to use interface
1. It is used to achieve abstraction
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling

Multiple inheritance in Java by interface
If a class implements multiple interfaces, or an interface extends multiple interfaces i.e known as multiple inheritence



Example

```
interface Printable{
void print();}
interface Showable{
void show();}
class A implements Printable,
Showable{
public void print(){
System.out.println("Hello");}
public void show(){
System.out.println("Welcome");}
public static void main(String args[]){
A a=new A();
a.print();a.show();}}
```

o/p - Hello
Welcome

Why Multiple inheritance is not supp. th. class in Java but it is possible by interface?
→ Multiple inheritance not supported in case of class because of ambiguity. But in case of interface, there is no ambiguity as implementation is provided by the implementation class.

```
interface Printable{
void print();}
interface Showable extends Printable{
void show();}
class A implements Showable{
public void print(){
System.out.println("Hello");}
public void show(){
System.out.println("Welcome");}
public static void main(String args[]){
A a=new A();
a.print();a.show();}}
```

o/p - Hello
Welcome

Packages	Interfaces
~~Interface~~	~~Packages~~
1. It is a group of classes and / or interface together.	1. Interface is a group of abstract method and constant fields.
2. Created using "Package" keyword	2. Created using "interface" keyword

3. Syntax:
package package-name;
public class class-name{
:
(body of class)
:
}
	3. Syntax:
	interface interface-name{
	variable declaration;
	method declaration;
	}

4. A package can be imported.	4. An interface can be extended by another interface and implemented by class but cannot be imported
5. Package can be imported using "import" keyword	5. It can be implemented using "implement" keyword

Class	Interface
1. It is ~~initial~~ instantiated to create objects.	1. It can never be instantiated as the methods are unable to perform any action on involving
2. keyword used to declare is class.	2. keyword used to declare is interface.
3. the members of a class can be private, public or protected	3. Members of interface are always public
4. methods of class are defined to perform a specific action	4. Methods in a interface are purely abstract.
5. A class can implement any no. of interfaces and can extend only one class	5. An interface can extend multiple interfaces but cannot implement any interface.
6. Class can have constructors to initialize the variable	6. An interface can never have a constructor

Abstract class	Interface
1. It can have both abstract and non-abstract method	1. Interface can have only abstract methods.
2. Doesn't support multiple inheritance.	2. Supports multiple inheritance
3. It can have final, non-final, static and non-static variables.	3. only have static & final variables.
4. It can provide implementation of interface	4. It can't provide the implementation of abstract class
5. abstract keyword is used to declare abstract class.	5. the interface keyword is used to declare interface.

6. Eg. public abstract class Shape{
public abstract void draw();
}
	6. Eg public interface Drawable{
	void draw();
	}

Inheritance

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of parent object

Idea is to create new classes that are built upon existing classes, so that you can reuse the methods and fields of parent class.

Inheritance used in Java due to
1. For method overriding (So runtime polymorphism can be achieved)
2. For code reusability

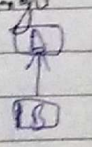
Syntax Inheritance

```
Class subclass-name extends Superclass-name
{
    // methods & fields
}
```

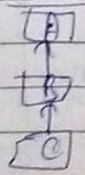
The extends keyword indicates that you are making a new class that derives from an existing class.

Types of inheritance

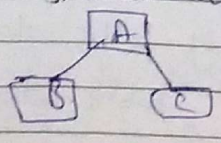
1) Single



2) Multilevel



3) Hierarchical



Difference between Method overloading & Method overriding

Method overloading	Method overriding
1. It is used to increase the readability of the program	1. It is used to provide the specific implementation of the method that is already provided by its super class.
2. It is performed within the class	2. It occurs always in derived class.
3. In this case Parameter must be different	3. In this case parameter must be same.
4. It is a example of compile-time polymorphism	4. It is example of run-time polymorphism.
5. Syntax (example)	5. Syntax (example)

Final variable, Final method and Final class or Various use of final keyword.

1. Final Variable: Use final to define constants
- If you want to make a local variable, class variable (static field), or instance variable (non-static field) constant, declare it final. A final variable may only be assigned to once and its value will not change and can help avoiding progr-

amming errors.

eg.
```
Class A {
  final int x = 10;
  void sum() {
    x = 20;
  }
  p.s.v.m (String a[]) {
    A a = new A();
    a.sum
  }
} // compile time error
```

2. Using final to prevent inheritance : Final class

A final class cannot be inherited

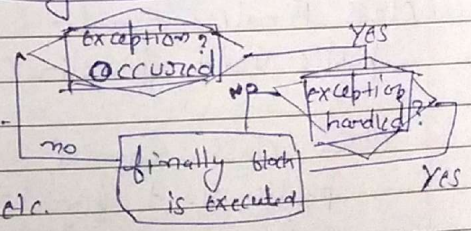
eg.
```
final class xyz { }
class ABC extends xyz {
  void demo() {
    s.o.p("A");
  }
  p.s.v.m (String a[]) {
    ABC a = new ABC();
    a.demo();
  }
}
```
o/p → exception: The type ABC cannot subclass the final class xyz

3. Final method : Using final to prevent overriding.

Methods declared as final cannot be overridden

```
class XYZ {
  final void demo() {
    s.o.p("XYZ class method");
  }
}

class ABC extends XYZ {
  void demo() {
    s.o.p("ABC class method");
  }
  p.s.v.m (String ar[]) {
    ABC a = new ABC();
    a.demo();
  }
} // compilation error
```

Finally Block



* It is a block that is used to execute important code such as closing conn., stream etc.
* It is always executed whether exception in handled or not
* It follow try or catch block

Syntax
```
class class_name {
  try { }
  catch( ) { }
  finally { }
}
```

Difference between final, finally, finalize

	Final	Finally	Finallize
1.	It is used to apply restrictions on class, method and variable - Final class can't inherit - Final method can't override - Final variable value can't changed	1. It is used to place imp. code, it will executed whether exception is handled or not.	1. It is used to perform clean up processing just before object is garbage collected.
2.	Final is a keyword	2. It is a block	3. It is a method

Java finalize ~~example~~ syntax

```
class Finalize_Example{
  public void finalize()

class class_name{
    public void finalize(){ }
    P.S.V.M (String args[]){
    class_name c = new class_name();
    c = null;
    System.gc();
    }
}
```

Static method

- If you apply static keyword with any method, it is known as static method
- It belongs to class rather than object of a class
- It can be invoked without the need for creating an instance of a class.
- It can access static data member and can change the value of it.

Syntax :
```
class class_name{
    static int method_name()
    p.s.v.m (String args[]){
        data_type var_name = classname.method_name();
    }
}
```

Static Block

- use to initialize the static data member
- execute before main method at the time of class loading.

~~Syntax~~ -
```
class class_name{
    static{ }
    p.s.v.m (String args[]){ }
}
```

Abstract method

If a class is abstract then the method inside the class may or may not abstract but where as if the method is abstract, then the corresponding class must be abstract. It has to be defined in subclass [compulsory for execution] opp to final method

Syntax for Abstract method:

```
Class Class_name1{
    abstract method_name();// no implementation
}                          no body

Class class_name2 extends 1 {
    abstract method_name();
    {// body
    }
}
```

Java Inner class

* Inner class is a part of nested class. Non-static nested classes are called inner class

* Java inner class or nested class is a class i.e declared inside the class or interface

Syntax

```
Class Java_outer_class{
  // code
  Class Java_Inner_Class{
    // code
  }
}
```

Advantages of Java inner class

1. It can access all the data members and methods of outer class including private.
2. It is used to develop more readable and maintainable code because it is logically group classes and interfaces in one place only.
3. Code optimization: It requires less code to write.

Types of Nested class

1. Non-static nested class [Inner class]

1) Member Inner Class

A class created within class and outside method

2) Anonymous Inner class

A class created for implementing interface and extending class. It's name is decided by the java compiler. It can be created by two ways

1. Interface
2. class

may be abstract or non-abstract

Syntax:

```
abstract class {
    abstract void method();
}

Anonymous Inner class {
    void method(){}
}
```

String

* Java string provides a lot of concepts that can be performed on a string such as compare, concat, equals, split, length, replace, compare to, substring etc.

* In Java string is basically an object that represents sequence of char values.

* An array of character works same as java String, For eg.

1. Char[] chr = {'j', 'a', 'v', 'a'};
2. String s = new String(chr);

* The java.lang.String class implements Serializable, comparable and char sequence interfaces.

* Some of the string class methods are

1. Char charAt(int ~~index~~ index)
   - returns char value for the particular index

2. String substring (int beginIndex)
   - returns substring for given begin index

3. String substring (int beginIndex, int endIndex)
   - returns substring for given begin index and end index

String Buffer Class

* ~~A class~~ It is used to create a mutable (A String that can be modified or changed is known as mutable string) string object

* StringBuffer and StringBuilder classes are used for creating mutable string.

Important methods of string Buffer class

1. Public synchronized StringBuffer append(String s)
   It is used to append the specified string with this string, eg. append(char), append(boolean)

2. Public synchronized StringBuffer insert(int offset, String s)
   It is used to insert the specified string with this string. eg. insert(int, boolean)

3. Public synchronized StringBuffer replace(int startIndex, int endIndex, String str)
   It is used to replace the string from specified startIndex and endIndex

Example

```
class A {
  p.s.v.m(String args[])
  {
    StringBuffer sb = new StringBuffer("Hello");
    sb.append("Java")
    S.o.p(sb) // prints Hello Java
  }
}
```

Page No.:
Date:

String Tokenizer

* The java.util.StringTokenizer class allows you to break a string into tokens. It is simple way to break string.

Constructors

1. StringTokenizer(String str)
   creates StringTokenizer with specified string
2. StringTokenizer(String str, String delim)
   creates ST with specified string and delimeter
3. StringTokenizer(String str, String delim, boolean return value)
   creates ST with specified string, delimeter and return value

   If return value is true, delimeter characters are considered to be tokens

   If false, delimeter characters serve to seperate tokens.

Methods of StringTokenizer class

1. boolean hasMoreTokens()
   - to check if there is more tokens available
2. String nextToken()
   - returns the next token from the StringTokenizer object

Example

```
import java.util.StringTokenizer;
public class simple{
    p.s.v.m(String args[]){
        StringTokenizer st = new StringTokenizer("my name is Hthen");
        while(st.hasMoreTokens()){
            S.o.b(st.nextToken());
        }
    }
}
```

o/p: my
name
is
Hthen

Difference between Applet and Application

Applet	Application
1. It implements html text	1. It cannot use HTML
2. It makes web pages.	2. It cannot make web pages
3. Applet programs are open by web browser	3. It doesn't need any web browser
4. Applet programs use applet package applet viewer (.java file)	4. Application use .java & class files
5. It is created by extending the java.applet.Applet	5. It is created by writing b.s.v.m(String[] args)
6. executed in restricted environment	6. Can access all resources of the computer

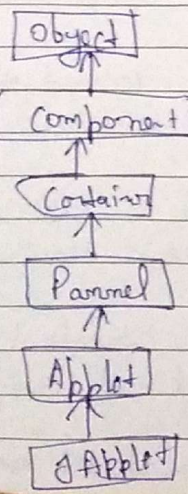
Java Applet

It is a special type of programme that is embedded in the webpage to generate the dynamic content, It runs inside the browser and works at client side.

Advantages
1. It works at client side so less response time.
2. Secured.
3. It can be executed by browsers running under many platforms including LINUX, windows, mac os etc.

Drawback
Plugin is required at client browser to execute applet

Hierarchy of Applet



Life cycle of Java applet

1. public void init(): used to initialize, invoked only once
2. public void start(): invoked after init(), used to start the applet
3. public void paint(): used to paint the applet, provides graphic class object that can be used for drawing oval, rectangle, arc etc.
4. public void stop(): used to stop the applet. invoke when applet is stoped or browser is minimised.
5. public void destroy(): used to destroy the applet. It is invoked only once.

Exception Handling

* Exception Handling is a mechanism to handle runtime errors such as classNotFoun, IO, SQL, Remote etc.

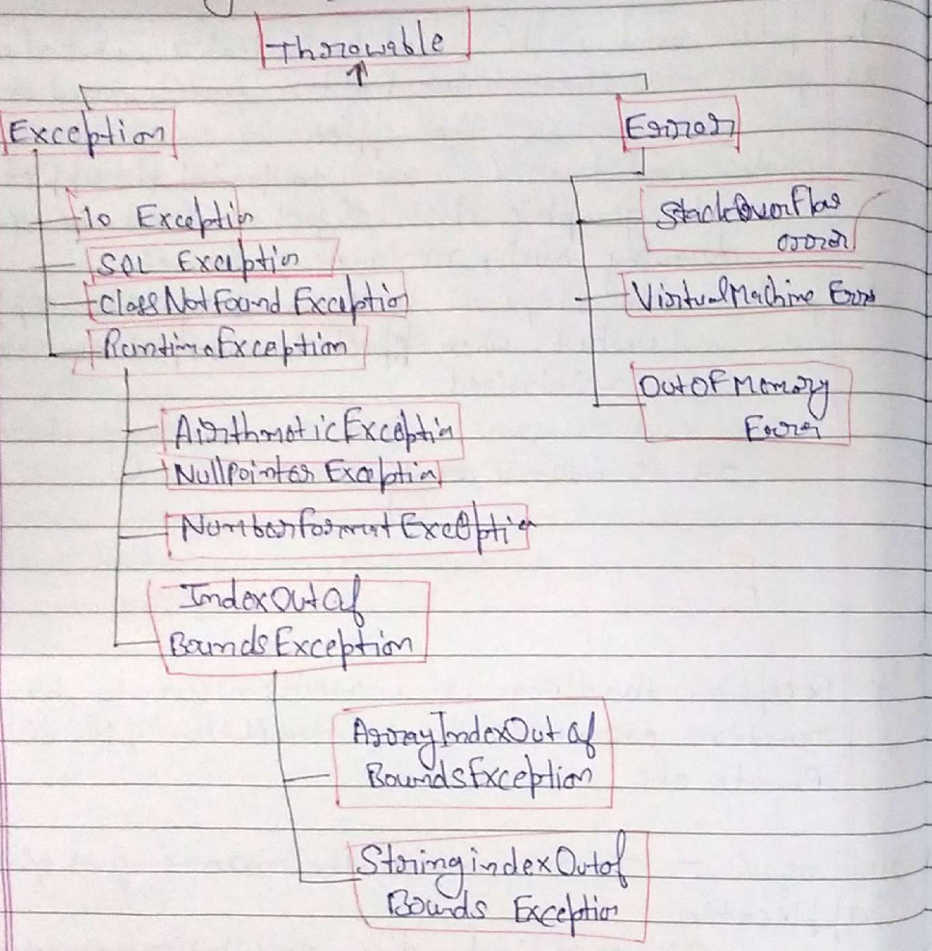
Advantage – To maintain the normal flow of the application
(in exception handing, combilation jumps over exception and executes the remaining code)

Types of Exceptions

1. Checked exception
2. Unchecked exception
   - Error

Page No.:
Date:

Hierarchy of Java Exception Classes



Chained exception

When the first exception can cause second exception

eg. ArithmeticException due to attempt of dividing a. no. by zero, but actually its a chained exception, the first exception is IOException due to which Arithmetic exception occurs

Page No.:
Date:

Difference between Throw and Throws key word

throw	throws
1. It is used to explicitly throw an exception.	1. It is used to declare an exception.
2. Checked exception cannot be propagated using throw only.	2. Checked exception can be propagated with throws
3. Throw is followed by an instance	3. Throws is followed by class
4. It is used within the method.	4. It is used with the method signature.
5. You cannot throw multiple exception	5. You can declare multiple exceptions eg. public void method() throws IOException, SQLException
6. Syntax: throw exception	6. Syntax: ~~method name~~ ~~method throws~~ method_name() throws exp1 { //method code }

Page No.:
Date:

Java try-catch

Java try block

- It is used to enclose the code that might throw an exception. It must be used within the method.
- It must be followed by either catch or finally block.

Syntax of java try-catch

```
try { // code that may throw exception }
catch (Exception-class-Name ref) { }
```

Syntax of try-finally

```
try { // code that may throw }
finally { }
```

Java - catch Block

- Used to handle exception, must be used after the try block.
- You can use multiple catch block with a single try.

Java - Multi catch Block

If we have to perform diff. tasks at occurrence of different Exceptions, we can use Java multi-catch block

```
public class MultipleCatchBlock {
    p.s.v.m (String a[]) {
        try {
            int a[] = new int[5];
            a[5] = 30/0;
        }

        catch (ArithmeticException e) {
            S.o.p ("task completed")
        }
        catch (ArrayIndexOutOfBoundsException e) {
            S.o.p ("task 2 comp.")
        }
    }
}
```

Try with resource [for file handling]

- Try-with-resources statement is a try statement that declares one or more resources.
- The resource is an object that must be closed after finishing the program
- The try-with-resources statement ensures that each resource is closed at the end of the statement execution

Final Rethrow

Difference between checked and unchecked exceptions

Checked	Unchecked
1. The compiler checks the checked exception	1. The compiler does not check the unchecked exception.
2. Except "RuntimeException" class, all the child classes of "Exception" class and "Error" class are "checked exception	2. "RuntimeException" class and its child classes are unchecked exception
3. If we do not handle the checked exception, then the compiler objects. Th	3. If we donot handle the unchecked exception, the compiler doesn't object
4. Program doesn't compile if there is an unhandled checked exception in the	4. The program compiles successfully even if there is an unhandled unchecked exception in the program code.

Java Threading

Multithreading in Java is a process of executing multiple threads simultaneously. Thread is basically a light-weight sub-process (smallest unit of processing). Multiprocessing and multithreading both are used to achive multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate seperate memory, so saves memory, and context-switching between the threads takes less time than process.

Advantages

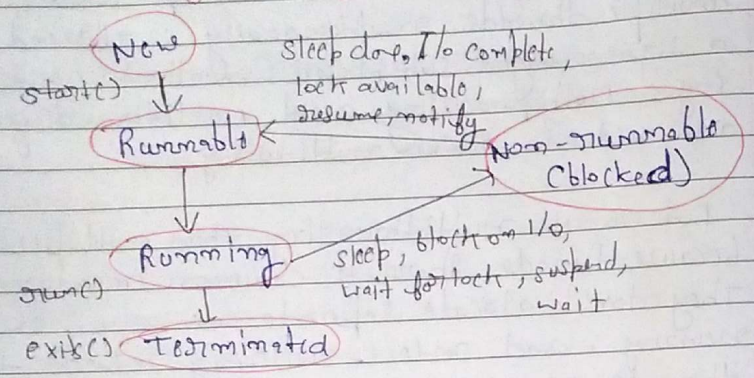
1. It doesn't block the user because threads are independent and you can perform multiple operations at same time.
2. You can perform many operations together so it saves time.
3. Threads are independent, so it doesn't affect any other threads if exception occurs in single thread.

Interthreading communication

The technique in which one thread produces some data and another thread is ready to consume.

Daemon thread – it only serves user threads for background support task
- low priority thread
- JVM terminate it automatically when all thread dies

Life cycle of a thread



1. New
The thread is in new state if you create an instance of Thread class but before the invocation of Start() method.

2. Runnable
Thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3. Running
The thread is in running state, if the thread scheduler has selected it.

4. Non-Runnable (Blocked)
This is the state when the thread is still alive, but is currently not eligible to run.

5. Terminated: A thread is in terminated or dead state when its run() method exists.

Thread Functions

isArrive()
determines if the thread is still running then join function wait for a thread to terminate.

getName()
takes the thread name

getPriority()
obtains the thread priority

yield()
causes the run time system to put the current thread to sleep and execute the next thread in the que.

Difference between Multitasking and Multithreading

Multitasking	Multithreading
1. Multitasking let CPU to execute multiple tasks at the same time.	1. Multithreading let CPU to execute multiple threads of a process simultaneously.
2. In this CPU switches between programs frequently.	2. In this CPU switches between the threads frequently.
3. In this system has to allocate memory and resources to each program.	3. In this system has to allocate memory to a process, multiple threads of the process shares the same memory and resources

Synchronization

Synchronization in java is the capability to control the acces of multiple threads to any shared resource.

It is better opinion where we want to allow only one thread to acess the shared resource

Sychronization is mainly used to
1. To prevent thread interference
2. To prevent consistency problem

Tykes of synchronization
1. Process Synchronization
2. Thread Synchronization

Two types of thread Synchronizatio are:
1. Mutual exclusive
   - Synchronized method
   - synchronized block
   - Static Synchronization
2. Cooperation (Inter-thread communication in Java)

Garbage Collection in Java

* It is the process of reclaiming the runtime unused memory automatically.

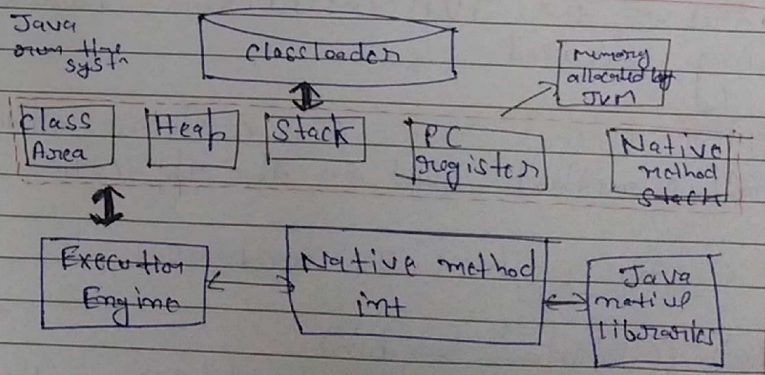
Advantages
1. It makes Java memory efficient because garbage collector removes the unreferenced objects from heap memory.
2. It is automatically done by garbage collector (a part of JVM)

finalize method already discussed

gc() method
- used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes

JVM Architecture

JVM (Java virtual machine) is an abstract machine. It is a specification that provides runtime environment in which Java bytecode can be executed. It is platform independent.



1. Class loader: It is a subsystem of JVM that is used to load class files.

Page No.:
Date:

2. Class (Method) Area : It stores per-class structures such as the run time constant pool, field and method data, the code for methods.

3. Heap- It is the runtime data area, in which objects are allocated.

4. Stack - Java stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return

Each thread has a private JVM stack, created at the same time as thread

A new frame is created, each time a method is invoked. A frame is destroyed when its method invocation is complete.

5. Program Register Counter - It contains the adres of the JVM instruction currently being executed

6. Native Method stack - It contains all the native methods used in the application

7. Execution engine : It contains

a) A virtual machine
b) Interpreter - Read bytecode stream then. Execute the instructions
c) Just-In-Time (JIT) compiler - It is used to improve the performance. JIT compiles byte code parts that have similar functionality at same time, and hence reduces the amount of time need for compilation.

Page No.:
Date:

Class File Format

* Compiled Java is typically distributed in a very compact format called class files. It is the assembler code for a virtual stack-oriented Java-friendly CPU chip.

* Class files are usually compressed and bundled into jar files. Each class file is marked with a class file format version, which lets you know which jdk it was compiled for

* A java class file is a file containing Java Bytecode that can be executed on the Virtual Machine (JVM). A Java class file is produced by a Java compiler from Java programming language source files (.java files) containing Java classes. If a source file has more than one class, each class is compiled into separate class file.

Security Promises of Java

* Security model - Sandbox model

* The language defines all primitives with a specific size, and all operations are defined to be in a specific order of execution. This code can be executed in diff. JVMs

* The language provide acess control functionality on variables & methods. This secures the program by restricting acess to its critical objects from untrusted code.

* The Java object encapsulation supports "programming by contract", which allows the reuse of code that already been tested.

* The java compiler does extensive type checking for type mistakes, thus it guarantees that the run-time data type variables are compatible.

* Classes & methods cannot be overridden which helps to protect the code from malicious attacks.

* The Java Garbage Collection mechanism provides a transparent storage allocation and recovering unused memory instead of deallocating the memory which prevents access to accidental and incorrect freeing of memory resulting in a JVM crash.

Thread [Continue]

Wait() & sleep()

* sleep(): It is a static method on thread class. It makes the current thread into the "Not Runnable" state for a specified amount of time. During this time, the thread keeps the lock (monitors) it has acquired

* Wait(): It is a method on object class. It makes the current thread into the "Not Runnable" state. Wait() is called on a object, not a thread. Before calling wait() method, the object should be synchronized, means the object should be inside "Synchronized block. The call to wait() releases the acquired lock.

		Sleep	Wait
1.	Class belongs	java.lang.Thread	java.lang.Object
2.	Context	Called from any context	only synchronized context.
3.	Locking	Does not release the locks for specified time or unt	Release the lock
4.	Wake up condition	When time expires due to interruption.	Awake by call to notify() or notifyAll() method
5.	Execution	Execution of sleep will pause the current running thread not the object on which it is called	Thread wait() continues till a specific condition holds true.

Page No. :
Date :

How to create Thread

There are two ways to create thread:-

1. By Extending Thread class

Thread Class:- It provide constructors and methods to create & perform operations on a thread. Thread class extends object class and implements Runnable interface.

Commonly used Constructor & methods of thread class

Constructors	Methods
1. Thread()	public void run()
2. Thread(String name)	public void start()
3. Thread(Runnable r)	public void sleep()
4. Thread	

Example

```
class Multi extends Thread{
public void run()
{
    S.o.p("thread is running...");
}
p.s.v.m (String args[])
{
    Multi t1 = new Multi();
    t1.start();
}
}
```

2. Runnable Interface

The Runnable interface should be implemented by any class whose instance are intented to be executed by a thread. Runnable Interface have only one method run().

public void run(): is used to perform action for a thread

Example

```
class Multi implements Runnable
{
    public void run()
    {
        S.o.p("thread is running...");
    }
    p.s.v.m (String args[])
    {
        Multi m1 = new Multi();
        Thread t1 = new Thread(m1)
        t1.start();
    }
}
```

Thread priority

* Each thread have a priority. Priorities are represented by a number between 1 and 10.
* In most cases, thread scheduler schedules the threads according to their priority (known as

preemptive scheduling), But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example

```
class TestMultiPriority extends Thread{
    public void run(){
        System.out.println("running thread name is:" + Thread.currentThread().getName());
        System.out.println("running thread priority is:" + Thread.currentThread().getPriority());
    }
    public static void main(String args[]){
        TestMultiPriority m1 = new TestMultiPriority();
        TestMultiPriority m2 = new TestMultiPriority();
        m1.setPriority(Thread.MIN_PRIORITY);
        m2.setPriority(Thread.MAX_PRIORITY);
        m1.start();
        m2.start();
    }
}
```

Thread Synchronization (Continue)

Concept of Lock in Java

Synchronization is built around an internal entity known as the lock or monitor. Every object has an lock associated with it

A thread that need access to an object's field has to acquire the object's lock before accessing them, and then release the lock when its done with them.

Synchronized method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task

Example

For Synchronized block

```
class Table{
    synchronized void printTable(int n){ //synchronized
        for(int i=1; i<=5; i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(Exception e){System.out.println(e);}
        }
    }
}
```

(object reference)

```
public class TestSynchronization
{
    p.s.v.m(String args[])
    {
        final Table obj = new Table();
        Thread t1 = new Thread()
        {
            public void run(){
                obj.printTable(5);
            }
        };

        Thread t2 = new Thread()
        {
            public void run(){
                ob.printTable(100)
            }
        };

        t1.start();
        t2.start();
    }
}
```

o/p -
5
10
15
20
25
100
200
300
400
500

Synchronized block in Java

* Synchronized block can be used to perform synchronization on any specific resource of the method.

* Suppose you have 50 lines of code in your method, but you want to synchronize, only 5 lines then you can use synchronized block

method in

* If we put all the code of a synchronized block, it will works same as synchronized method

Points to remember

1. Synchronized block is used to lock an object for any shared resource
2. Scope of synchronized block is smaller than the method

Syntax

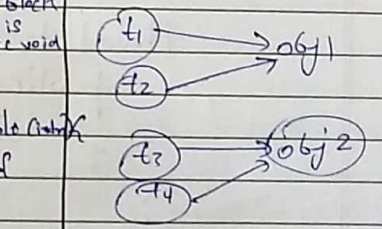
```
synchronized (object reference expression) {
    // code block
}
```

Static Synchronization

If you make any static method as synchronized, the lock will be on the class not on object

Example [for block - synchronized block will be added & method is static void]

```
class Table{
    synchronized static void PrintTable(int n){
        for(int i=1; i<=10; i++){
            s.o.p(n*i);
            try{
                Thread.sleep(100);
            }
            catch(Exception e){}
        }
    }
}
```

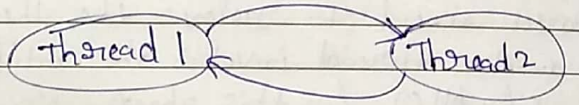


In static sync. no interference is possible b/w t1/t2 and t3/t4.

```
class MyThread1 extends Thread{
    public void run(){
        Table.printTable(1);

public class TestSynchronization{
    p.s.v.m(String[] args){

        Thread t1 = new Thread(){
            public void run(){
                Table.printTable(1)
        };

        Thread t2 = new Thread(){
            public void run(){
                Table.printTable(10)
            }
        };

        Thread t3 = new Thread(){
            public void run(){
                Table.printTable(100)
            }
        };
        Thread t4 = new Thread(){
            public void run(){
                Table.printTable(1000);
            }
        };
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }
}
```

Deadlock

It is a part of multithreading, it can occur in a situation when a thread is waiting for an object lock, that is acquired by another ~~object~~ thread and second thread is waiting for an object lock that is acquired by first thread. Since both the object is waiting for each other to release the lock ~~that is acquired~~, the condition is called deadlock.



Example

```
public class TestDeadlock{
    p.s.v.m(String[] args){
        final String resource1 = "yogesh";
        final String resource2 = "Tarachay";

// t1 tries to lock resource1 then resource2
        Thread t1 = new Thread(){
            public void run(){
                synchronized(resource1){
                    s.o.p(" Thread1: locked resource1");

                    try{Thread.sleep(100);}catch(Exception e){}
                synchronized(resource2){
                    s.o.p(" Thread1 : locked resource2");
                }
            }
        };
// t2 tries to resource2 then resource1
        Thread t2 = new Thread(){
            similar code
        };
        t1.start();
        t2.start();
    }
}
```

Page No. :
Date :

Inter-thread Communication in Java

It is a mech. in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed it. It is implemented by following method of object class.

1. wait() method
It causes thread to release the the lock and wait until another thread invokes the notify() method or notifyAll() for this object, or a specified amount of time has elapsed.

Method
public final void wait() throws InterruptedException
- waits until object is notified

public final void wait(long timeout) throws InterruptedException
- waits for the specified amount of time

2. notify() method
If any threads are waiting on this object, one of them is choosen to be awakened, the choice is arbitary

Syntax
public final void notify()

3. notifyAll() method
wakes up all threads that are waiting on this object's monitor

Syntax
public final void notifyAll()

Example

```
class Customer{
  int amount=10000;

  synchronized void withdraw(int amount){
    s.o.p("going to withdraw...");

    if(this.amount < amount){
      s.o.p("Less balance; waiting for deposit...");
      try{wait();}catch(Exception e){}
    }

    this.amount-=amount;
    S.o.p("withdraw completed...");
  }

  synchronized void deposit(int amount){
    s.o.p("going to deposit");
    this.amount+=amount;
    s.o.p("deposit completed...");
    notify();
  }
}

Class Test{
  p.s.v.m(String args[]){
    final Customer c=new Customer();
    new Thread(){
      public void run(){c.withdraw(15000);}
    }.start();
    new Thread(){
      public void run(){c.deposit(10000);}
    }.start();
  }
}
```

o/p
going to withdraw
less balance, waiting for deposit
going to deposit
deposit completed...
withdraw completed...

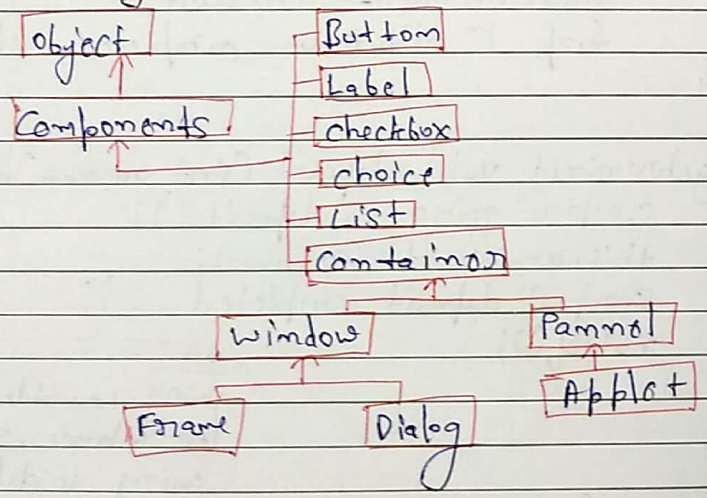
Page No. :
Date :

Java AWT Components

* Java AWT (Abstract window Toolkit) is an API to develop GUI or window based applications in Java.

* JAVA AWT components are platform-dependent ie. components displayed according to the view of operating system, its components are using the resources of OS.

Hierarchy



container - The container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends container. The classes that extends container class are known as container such as Frame, Dialog and Panel.

window - The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window menu

Pannel - It doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame - It contain tittle bar and can have menu bars. It can have other components like button textfield etc.

Useful Methods of Component class

public void add(Component c) - inserts a component on this component.

public void setSize(int width, int height) - Sets the size (width and height) of the component

public void setLayout(LayoutManager m) - defines the layout manager for the component

public void SetVisible(boolean status) - changes the visibility of the component, by default false.

Example

To create simple example of AWT, you need a frame in AWT.

- By extending Frame class (inheritance)
- By creating the object of Frame class (Association)

1. By Inheritance

```
import java.awt.*;
class First extends Frame{
  First(){
    Button b = new Button("click me");
    b.setBounds(30,100,80,30); // position
    add(b); // adding button into frame
    setSize(300,300); // frame size
    setLayout(null); // no layout manager
    setVisible(true); // now frame will be visible
  }
  p.s.v.m(String args[]){
    First f = new First();
  }
}
```

2. By Association

```
import java.awt.*;
class First{
  First(){
    Frame f = new Frame();
    Button b = new Button("click me");
    b.setBounds();
    f.add(b)
    // Same code body
    f.setSize(300,300)
    f.setLayout(null)
    f.setVisible(true)
  }
```

How to close AWT window

We can close the AWT window or frame by calling dispose() or system.exit() inside windowClosing method. The windowClosing() method is found in WindowListener interface and WindowAdapter class.

Different ways to override windowClosing() method

1. By anonymous class
2. By inheriting WindowAdapter class.
3. By implementing WindowListener interface

Close AWT window Example by Anonymous class

```
import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowExample extends Frame{
    WindowExample(){
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                dispose();
            }
        });
        setSize(400,400)
        setLayout(null)
        setVisible(true)
    }
p.s.v.m(String[] args){
    new WindowExample();
}
}
```

Page No. :
Date :

Difference Between Get request method and Post method.

Get	Post
1. In case of Get request, only limited data can be sent because data is sent in header.	1. In case of Post request, large amount of data can be sent because data is sent in body
2. not secured as data is exposed in URL bar.	2. secured, because data is not exposed in URL
3. It can be bookmarked	3. It cannot be bookmarked
4. It is idempotent, i.e. second request will be ignored until response of first request is delivered.	4. not - idempotent
5. It is more efficient & used more than Post	5. less efficient and used less than Get

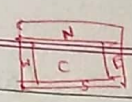
Layout Manager Interface Default Layout

* The LayoutManagers are used to arrange component in a particular manner. ~~It is an~~ LayoutManager is an interface that is implemented by all the classes of layout mangers.

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.GridBagLayout
5. java.awt.CardLayout

Page No. :
Date :

Java Border Layout



used to arrange the components in five regions: north, south, east, west and centre. Each region may contain one component only. It is default layout of frame and window.

eg. public static final int NORTH

Construction used — BorderLayout() JBorderLayout(int hgap, int vgap)

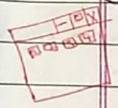
creates borderlayout but with no gaps between components

creates border layout with given horizontal & vertical gaps

Java GridLayout - used to arrange components in rectangular grid

Constructors - GridLayout()
GridLayout(int rows, int columns)
GridLayout(int rows, int columns, int hgap, int vgap)

Java FlowLayout - used to arrange components in a line, default layout of applet or panmel



Constructors - FlowLayout()
FlowLayout(int align): gives allignment
FlowLayout(int align, int hgap, int vgap)

Java CardLayout = This class manages the components in such a manner that only one component is visible at a time. It treats each component as a card.

Constructors - CardLayout()
CardLayout(int hgap, int vgap)

Methods - public void next(Container parent) - is to flip to next card with the given name
previous
first
last
show

Example (FlowLayout)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CardLayoutExample extends JFrame
public class MyFlowLayout{ JFrame f; MyFlowLayout()
    f = new JFrame();

    JButton b1 = new JButton("1")
    JButton b2 = new JButton("2")
    JButton b3 = new JButton("3")

    f.add(b1); f.add(b2); f.add(b3)
    f.setLayout(new FlowLayout(FlowLayout.RIGHT));

    f.setSize(300,300)
    f.setVisible(true);
}
p.s.v.m(String[] args){
    new MyFlowLayout();
}
}
```

Java GridBagLayout - It is used to align components vertically, horizontally or along their baselines.

Components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells.

Event Handling

Changing the state of an object is known as event. eg. click on button, dragging mouse etc. The java.awt.event package provides many event classes and listener interface for event handling.

Java event classes & Listener Interface

Event class	Listener Interface
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
WindowEvent	WindowListener
TextEvent	TextListener

Steps to perform Event handling
- Register the component with the listner

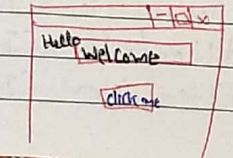
Many classes provided for the registration method.

1. Button - public void addActionListener(ActionListener a){}
2. MenuItem - public void addActionListener(ActionListener a){}
3. CheckBox - public void addItemListener(ItemListener a){}
4. Choice - public void addItemListener(ItemListener a){}

Example

```
import java.awt.*;
import java.awt.event*;

Class Event extends Frame {
    TextField tf;
    Event() {
        tf = new TextField();
        tf.setBounds(60,50,170,20);
        Button b = new Button("click me")
        b.setBounds(50,120,80,30);

        b.addActionListener(new ActionListener() {
            public void actionPerformed() {
                tf.setText("hello");
            }
        });

        add(b); add(tf);
        setSize(300,300)
        setLayout(null)
        Set Visible(true);
    }

    p sv m (String args[]) {
        new Event();
    }
}
```

o/p

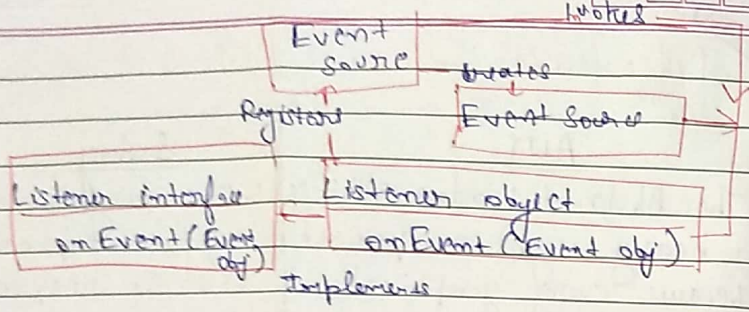


Difference between AWT and Swing

AWT	Swing
1. The Abstract Window Toolkit is a heavy weight component because every graphical unit will invoke the native methods	1. The Swing is a light weight component because it is the responsiblity of JVM to invoke the native methods.
2. The look & feel of AWT depends upon platform	2. As Swing is based on (model view Controller pattern), the look & feel of swing components is independent of hardware & operating system.
3. occupies more memory space.	3. occupies less memory space.
4. AWT is less powerful than String	4. Swing is extension of Awt and many drawbacks of AWT are removed in Swing
5. AWT uses native libraries	5. Swing components have been completly written in Java language

Event Delegation model

- Based on Event Source & Event Listeners (Event classes & Objects)
- Event Listener is an object that recieves the messages / events
- The event source is any object which creates the message / event.



Three participants in event delegation model in Java

Event Source - The class which broadcasts the event
Event Listeners - The class which recieve notification of events
Event object - The class which describes the event

Adapter Classes

(extension of interface)

- To stop overridding all methods → define, & Blank
- Used to simplify the process of event handling
- Provide empty implementation of all methods in an event listner interface
- Define a new class to act as event listner by extending one of the adapter classes and implementing only those methods that you want to use in your code.

Example

```
class adapter1 extend Frame
{
    Frame f
    adapter1()
    {
```

Mouse motionlistner → mousemoved, mouse drag

```
        f = new frame(" ")
        f.addMouseMotionListner();
    }

    class m1 extends MouseMotionAdapter
    {
        public void mouseMoved(MouseEvent me)
        {
            = → functionality
        }
    }
}
```

JSP Implicit objects

Java implicit objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. objects are also called pre-defined variables

JSP Supports nine implicit objects, which are listed below

1. The Request object:- It is an instance of javax.servlet.http.HttpServletRequest object
   - Each time a client req. a page, the JSP engine creates a new object to represent that req.
2. The Response object:- instance of javax.servlet.http.HttpServletResponse object
   - creates obj to represent response of client.
3. The out object - implicit object, javax.servlet.jsp.JspWriter object
   - used to send content in a response

4. The Session object :- javax.servlet.http.HttpSession
   - behave same as session objects behave under java servlet

5. The Config object :- javax.servlet.http.HttpConfig
   - It is a direct wraper around Servlet Config object

6. The page object :- It is an actual reference to the instance of the page.
   - It is an object that represents the entire JSP page.

Java I/O [File Handling]

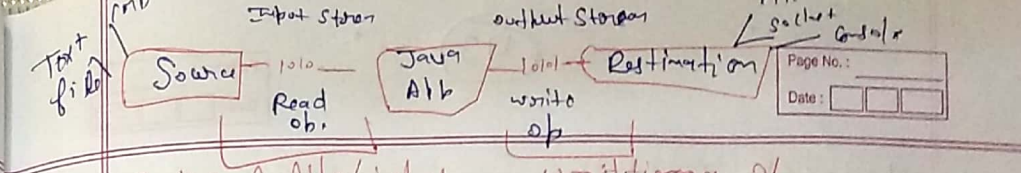
- java.io package is used
- use the concept of Stream to make I/O operation fast

Stream is a sequence of data
→ composed of bytes

System.out — o/p Stream
System.in — i/p Stream
System.err — Error Stream

output Stream class (write byte)
- void write(int) throws IOException
- void write(byte[]) throws IOException
- void flush() throws IOException
- void close() throws IOException

Input Stream class (read next byte)
- int read() throws IOException
- int available() throws IOException
- void close() throws IOException



Java Buffered output Stream class

- used internal buffer to store data
- Fast & more efficient

Example

```
import java.io.*;
class file
{
    p.s.v.m (String[] args)
    {
        FileOutputStream fout = new FileOutputStream(fi
        BufferedOutputStream b1 = new BufferedOutputStream
        String S = "EEC classes";                    (fout)
        byte b[] = S.getBytes();
        b1.write(b);
        b1.close();
        fout.close();
    }
}
```

Java Buffered Input Stream Class

- used to read information from stream
- use buffer mechanism to make performance fast

Example

```
Class file2 {
    psvm (String[] args) {
        FileInputStream fin = new FileInputStream("fi.txt")
        BufferedInputStream b2 = new BufferedInputStream(fi
        int ch;
        while((ch=b2.read())!=-1){System.out.print((char)ch);}
    }
}
```

PrintStream Class

→ provides methods to write data to another stream.
→ Automatically flushes the data

Example

```
import java.io.*;
Class file1
{
    p.s.v.m (String[] args) throws Exeption
    {
        FileOutputStream fout = new FileOutputStream("
        PrintStream p1 = new PrintStream (fout) f1.txt
        p1.println (400);
        p1.println ("EEC");
        p1.close();
        fout.close();
    }
}
```

Example of formatting using printf() method

```
Class file2 {
    p.s.v.m (String[] ar)
    {
        int i=5
        S.o.p ("%d", i);
    }
}
```

Java Data OutputStream Class

- This class allows an application to write primitive java data types to the output stream in a machine independent way
- generally uses data output stream to write data

Example

```
import java.io*;
public class Example {
    p.s.v.m (String[] args) throws IOExceptions
    FileOutputStream fout = new FileOutputStream (f1.txt)
    DataOutputStream d = new DataOutputStream (fout)
    d.writeInt(65);
    d.flush
    d.close
    S.o.p ("success");
}
}
```

Java Data Input Stream Class

- To read primitive data in a input stream in a machine independent way
- generally use data input stream to read data

Same

```
int count = fin.available();
byte[] ary = new byte[count];
d.read(ary)
for (byte bt : ary) {
    char k = (char) bt
    S.o.p (k+"-");
}
}
```

Java FilterWriter

- Abstract class used to write filtered character stream

Methods used - FileReader(), BufferedReader(), ~~Filter Custom Filter Write~~, FileWriter, extend FilterWriter, super()

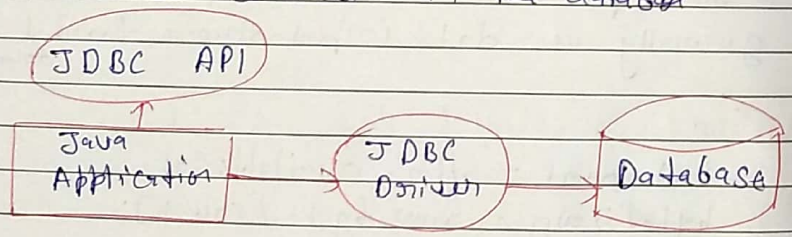
Java FilterReader

- abstract class for reading filtered character stream

Methods:- extends FilterReader, Reader(), super()

JDBC

* Java JDBC is a java API to connect & execute query with the database. JDBC Api uses jdbc drivers to connect with the database



* Before JDBC, ODBC API was the database API to connect & execute query with the database But, ODBC API uses ODBC drivers which is written in C language (i.e. platform dependent and unsecured), thus Java define its own API (JDBC API) that uses JDBC drivers (written in JAVA language)

* API (Application programming Interface) is a document that contains description of all the features of a product / software.

It represents classes and Interfaces that software programs can follow to communicate with each other.

An API can be created for applications, libraries, operating System etc.

JDBC Driver

JDBC driver is a software component that enables java application to interact with the database

There are 4 types of JDBC drivers:-

1. JDBC - ODBC bridge driver
   - uses ODBC driver to connect to the DB.
   - It converts JDBC method calls to ODBC function calls.
   - now discouraged due to thin driver
   - Easy to use, easily connected to any DB
   - Performance degrade due to call conversion, also it needs to be installed at the client machine
2. Native API driver (not written entirely in JAVA)
   - uses client-side libraries of the DB
   - JDBC method calls → native calls of DB API
   - Better than JDBC - ODBC driver.
   - Native drivers needs to be installed on each client machine.

3. Network Protocol driver (fully written in JAVA)
- uses application server that converts JDBC calls directly / indirectly into the vendor-specific DB protocol.
- No client side library is required
- Maintainence is costly
- Network support is required on client machine.

4. Thin Driver (fully written in JAVA)
- Converts JDBC calls directly into the vendor-specific DB protocol.
- Better than all other drivers
- No software is required at client side or server side
- Drivers depend on the Database.

5 steps to connect to the DB in Java

1. Register the driver class

Syntax - public static void forName(String className) throws ClassNotFoundException

Example - Class.forName("oracle.jdbc.driver.OracleDriver");

2. Create the Connection object (to establish conn.)

Syntax - public static Connection getConnection(String url) throws SQLException

Example - Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost", "system", "password");

3. Create the statement object
- to create statement, i.e. object of statement is responsible to execute queries with the DB

Syntax: public Statement createStatement() throws SQLException

Example: Statement s = con.createStatement();

4. Execute Query
- executeQuery() method of statement interface is used to execute queries to the DB
- returns the object of ResultSet that can be used to get all the records of a table.

Syntax - public ResultSet executeQuery(String sql) throws SQLException

Example - ResultSet rs = stmt.executeQuery("select * from emp");
while(rs.next()){
s.o.println(rs.getInt(1)+" "+rs.getString(2));
}

5. Close the connection object (close() method of Connection interface used to close connection)

Syntax - public void close() throws SQLException

Example - con.close();

Page No.:
Date:

Connectivity with oracle

1. Driver Class: oracle.jdbc.driver.OracleDriver
2. Connection URL:
   jdbc:oracle:thin:@localhost:1521:xe
3. Username: By default is "system"
4. Password: given by user at the time of installation

Create a table in oracle Data Base
Create table emp (id number(10), name varchar(40), age(6));

Example

```
import java.sql.*;
Class Oracle con {
    p.s.v.m (String args[]) {
        try {

            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe", "system",
                                                   "oracle");
            Statement s = con.createStatement();

            ResultSet rs = s.executeQuery("select * from emp");
            while(rs.next())
            S.o.p(rs.getInt(1)+" "+rs.getString(2)+" "+rs.
                                                   getString(3));
            con.close();
        } Catch(Exception e) {S.o.p(e)}
}}
```

Page No.:
Date:

Connectivity with MySQL

1. Driver Class: com.mysql.jdbc.Driver
2. Connection URL: jdbc:mysql://localhost:3306/xyz
3. Username: root
4. Password: root

Example Create DB

```
Create database xyz
use xyz
Create table emp (id int(10), name varchar(40),
                                    age int(6));
```

```
impor java.sql.*;
Class mysqlcon {
    p.s.v.m (String args[]) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/xyz", "root", "root");

            Statement s = con.createStatement();

            ResultSet rs = s.executeQuery("Select * from emp");
```

→ Same as

Connectivity with MS-Access

We make a connection using JDBC to a MS-Access data base. The connection is made with help of a JDBC odBC driver

Creating a DB

1. Open MS Access and select Blank DB option and give the DB name as File name option
2. Create a table & insert your data into the table
3. Save table with desired name

Creating DSN of your DB

4. Control panel → Administrative tools
5. Data Source (ODBC) → System DSN
6. add → select MS-Access Driver (*.mdb, *.accdb) → Finish
7. Make Desired Data Source name → Select option → MYDSN
8. Select Data Source File → ok → Create → Finish
9. Program Code

```
Class.forName("sun.jdbc.odbc.jdbcodbcDriver");
Connection con = DriverManager.getConnection
                 (url, userid, password)
Statement s = con.createStatement()

ResultSet rs = s.executeQuery("Select * from tbl");
while (rs.next())
s.o.p(rs.getInt(1)+" "+rs.getString(2))
con.close
```

Serialization in Java

It is a mechanism of writing the state of an object into a byte stream.

Mainly used in Hibernate, RMI, JPA, EJB and JMS technologies

Serialisation: object → Stream; Deserialisation: Stream → object

Advantage - It is mainly used to travel object's state on the network (known as marshaling)

Example

```
import java.io.*;
class Persist {
    p.s.v.m (String args[]) throws Exception {
        Student s1 = new Student(211, "ravi")

        FileOutputStream fout = new FileOutputStream("f.txt")
        ObjectOutputStream out = new objectoutputstream(fout)

        out.writeObject(s1)
        out.flush()
        S.o.p("success")
    }
}
```

(state of object saved; provides functionality of serialization)

Networking

1. Socket Programming
   - It is used for communication between the applications running on different JRE.
   - It can be connection-oriented or connection-less

* Socket and ServerSocket classes are used for connection oriented socket programming

DatagramSocket & DatagramPacket Classes are used for connectionless socket-programming

The client in socket programming must know two information
1. IP Adress of server
2. Port number

Socket Class
A socket is simply an end point for communi-cations between the machines. The Socket class can be used to create a socket.

Method	Description
1. public InputStream getInputStream()	returns inputstream attached with socket
2. public outputStream getOutputStream()	" outputstream
3. public Synchronized Void close	closes the socket

Server Socket Class
SS class can be used to create a Server Socket This object is used to establish communication with the clients.

Method	Description
1) public socket accept()	returns socket and establish comm. b/w server & client
2) public synchronized void close()	closes the Server Socket

2. Development of Client-Server applications

- Use of Sockets
- Socket is end-point for comm. b/w machines over network
- Two programs can be run on the same computer,
- Two distinguish numbering convention - Port no. used
- IP Adress of Server & Port no.

* A network comm. is initiated by client when it creates a socket for the communication with the server
- Server receives comm. req. over its specific server port, it creates new socket for it and binds a port no. to it.
- Server sends the new port no. to inform him that comm. is established for accept() server new port no. → client
- The server then reads and writes messages on established connection (on new port) with the accepted client.

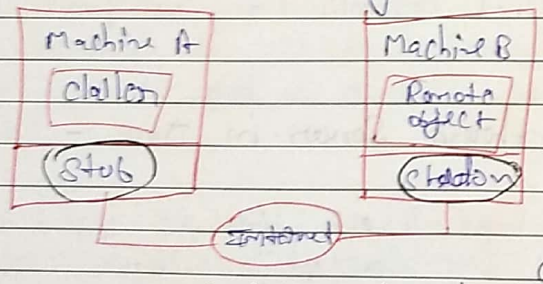
3. Multi-threaded Server in Java - Pg 207 Ankit NTP-1-A

Advantages - 1. Less time is spent outside accept() call.
2. Long running client req. do not block the whole server.
3. Perform many functions at once, saves lot of time

Disadvantages - 1. Creating multi-thread prog. not easy. 2. Difficult to debug

RMI [Remote Method Invocation]

* RMI is an API that provides a mechanism to create distributed application in Java.
* RMI allows an object to invoke methods on an object running in another JVM.
* RMI provides remote communication between the applications using two objects Stub & Skeleton
* A remote object is an object whose method can be invoked from another JVM
* Stub is an object acts as an gateway for the client side. All the outgoing requests are routed through it
* Skeleton is an object acts as an gateway for the skeleton. All incoming req. are routed thro it



CRMI had all the

Requirements for distributed application:-

1. Application need to locate the remote method
2. It needs to provide comm. with remote object.
3. It needs to load the class definitions for the objects

JAVA RMI Example / steps to write RMI Program

1. Create the remote interface.
2. Provide the ~~interface~~ implementation of the remote interface.
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool.
4. Start the registery service by rmiregistry tool.
5. Create and start the remote application
6. Create and start the client application.

JNI [Java Native Interface]

- An interface that allows JAVA to interact with code written in another language.

- Advantages:
  1. Code reusability (mostly C/C++)
  2. Performace as Native code is used to be upto 20 times faster than Java, when running in interpreted mode
  3. Modern JIT compilers (Hotspot) make this as advantage.
  4. Allow Java to tap into low level O/S, H/W routines

- JNI code is not portable
- JNI can invoke Java code from within native written applications of languages C/C++
  eg. Java command line utility

JNI components

1. javah - JDK tool that builds C-style header files
2. jni.h - c/c++ header file included with the JDK that maps Java types to their native counterparts

[javah automatically includes this file in the application header file]

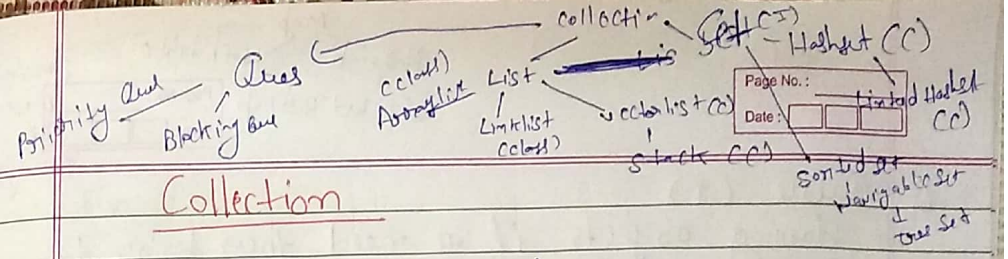
JNI Development (Java)

1. Create a Java class with native methods
   public native void sayHi(String who, int times);
2. Load library which implements the method
   System.loadLibrary("HelloImpl");
3. Invoke native method from Java

Example

```
package Example
public class Hello {
    static {
        System.loadLibrary("HelloImpl");
    }
    public static void main(String[] args) {
        Hello h = new Hello();
        h.sayHi(args[0], Integer.parseInt(args[1]));
    }
}
```

(1) (2) (3)



Collection

- It is a group of individual objects as a single entity.
- Collection Framework defines several classes & interfaces which can be used to represent a group of objects as a single entity.

Elements of Collection Framework

1. Collection (Interface) (Represent a grp. of objects as a single entity)
   - defines most of the methods
2. List (Interface)
   - List is a child of Collection interface
   - Insertion order is preserved, duplicates are allowed
3. Set (Interface)
   - same as List but duplicates are not allowed
   - and insertion order is not preserved
   - child of Collection interface
4. Sorted Set (Interface)
   - child of Set
   - according to some sorting order
5. Navigable Set (I)
   - used to provide several methods for Navigable process

Map → HashMap, Hashtable, SortedMap
HashMap → LinkedHashMap

6. Queue (I)
- storing objects if we need them prior to processing

7. Map (I)
→ to represent key value pairs (objects)
- duplicate keys are not allowed
- values can be duplicated

8. Sorted Map (I)
- sorting order
- key based sorting

9. Navigable Map (I)
- child interface of SortedMap
- Navigable process

Vector Class
- Implement array of objects
- It contains the components that can accessed using an integral index
- Size of the vector can grow or shrink as needed to accommodate adding or removing, adding/removing items after the vector has been created
- Each vector tries to optimize storage management by maintaining a capacity and capacity increment

Example

```
package vectorExample;
import java.util.*;
public class MyVectorCollection {
    p.s.v.m (String args[]){
        Vector<String> vct = new Vector<String>();

        vct.add("First");
        vct.add("Second");
        vct.add("Third");
        vct.add("Realm");
        S.op("Actual Vector" +vct);

        List<String> list = new ArrayList<String>();

        list.add("one");
        list.add("two");
        vct.add(list);
        S.op("After copy:" +vct);
    }
}
```

o/p - Actual Vector: [First, Secon, Third, Realm]
After copy: [[1], one, two]

Iterator
- when for-each loop is not available, and and explicit Iterator is needed, then Iteration over a collection may be done with a: while loop / for loop

both have diff. advantages:-

1. while loop is considerably more reliable
2. for loop minises the scope of Iterator to the loop itself

Scanned by CamScanner