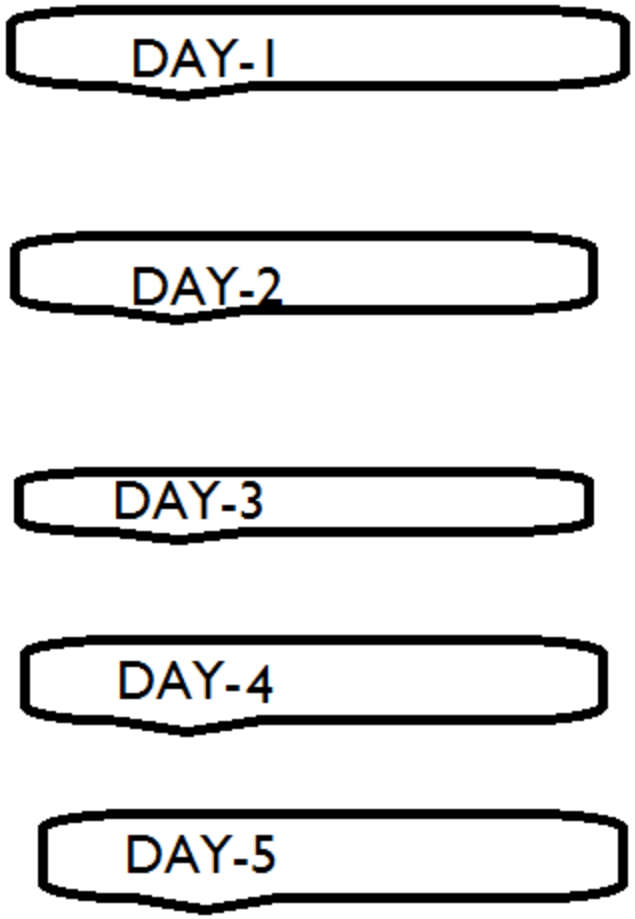
Core Java

Rajeev Gupta
MTech CS

Contents at a glance

- Java An Introduction
- Basic Java Constructs and Data Types and
- OO Introduction
- Class object and related topics
- Inheritance
- Final ,Abstract Classes and interface
- String class, immutability, inner classes
- Exceptions Handling, assertions
- Streams and Input/output Files, serializati
- Java Thread
- Java Collections



Day-1

Session-1

- Introduction to Java
- What is Java,Why Java?
- JDK,JRE and JVM discussion
- Installing JDK, eclipse,javadoc
- Hello World application
- Procedural Programming
  - If..else,switch, looping
  - Array
  - Examples

- Object Oriented
  - Object, class, Encapsulation,
  - Abstraction,
  - Inheritence,
  - Polymorphism,message passing

    (concepts only, implementation on day 2)

rgupta.mtech@gmail.com

Day-2

Session-1

Class and objects
Creating Classes
Implementing Methods
Object: state, identity and behaviour
Constructors: default, parameterized and copy
Instance variable, static variable and local variable
Initialization block
Static method and applications
Packages and visibility modifier: public, protected, private and default

rgupta.mtech@gmail.com

Day-3

Session-1

- String class: Immutability and usages, stringbuilder and stringbuffer

- Wrapper classes and usages
- Java 5 Language Features (I):

- AutoBoxing and Unboxing, Enhanced For loop, Varargs, Static Import, Enums
-
- Inner classes
- Regular inner class, method local inner class, annonymous inner class

rgupta.mtech@gmail.com

Day-4

Session-1

IO

Char and byte oriented streams

BufferedReader and BufferedWriter

File handling

Object Serialization and IO

[ObjectInputStream / ObjectOutputStream]

rgupta.mtech@gmail.com

Day-5

Session-1

Collections Framework
List, Map, Set usages introduction
Usages ArrayList, LinkedList, HashMap, TreeMap, HashSet
Comparable, Comparator interface implementation
User define key in HashMap

rgupta.mtech@gmail.com

DAY-1

rgupta.mtech@gmail.com

Session-1

- Introduction to Java
  - What is Java,Why Java?
  - JDK,JRE and JVM discussion
  - Installing JDK, eclipse
- Hello World application
- Procedural Programming
  - If..else,switch, looping
  - Array
- Examples
-

rgupta.mtech@gmail.com

What is Java?

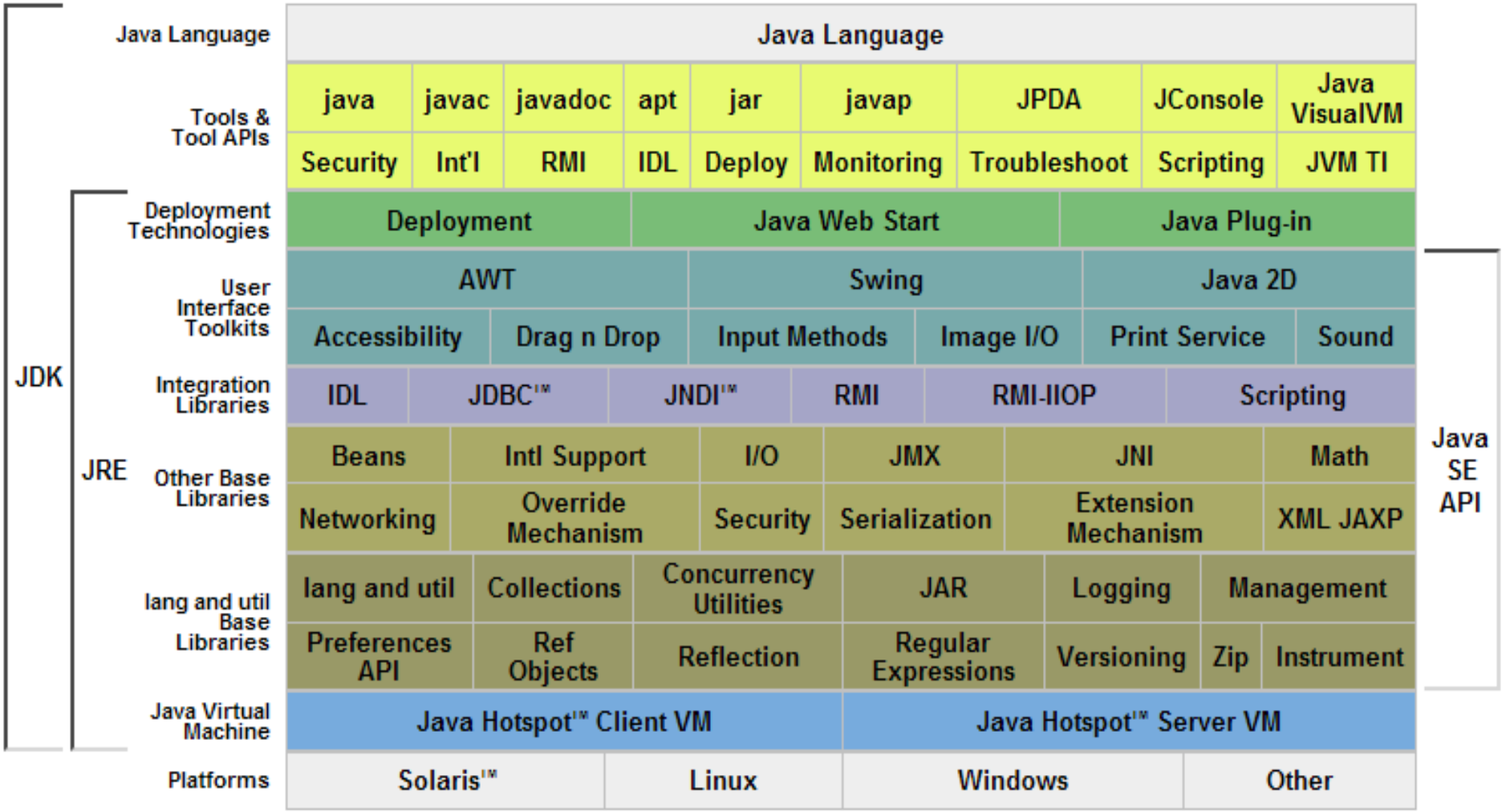
- Java=OOPL+JVM+lib
- How Java is different then C++?

rgupta.mtech@gmail.com

How Sun define Java?

- Simple
- Object Oriented
- Portable
- Architecture Neutral
- Distributed
- Multi-threaded
- Robust, Secure/Safe
- Interpreted
- High Performance
- Dynamic programming language/platform.

Java Platform



rgupta.mtech@gmail.com

Versions

- JDK 1.0 (1995)
- JDK 1.1 (1997)
- J2SE 1.2 (1998)) Playground
- J2SE 1.3 (2000) Kestrel
- J2SE 1.4 (2002) Merlin
- J2SE 5.0 (2004) Tiger
- Java SE 6 (2006) Mustang
- Java SE 7 (2011) Dolphin

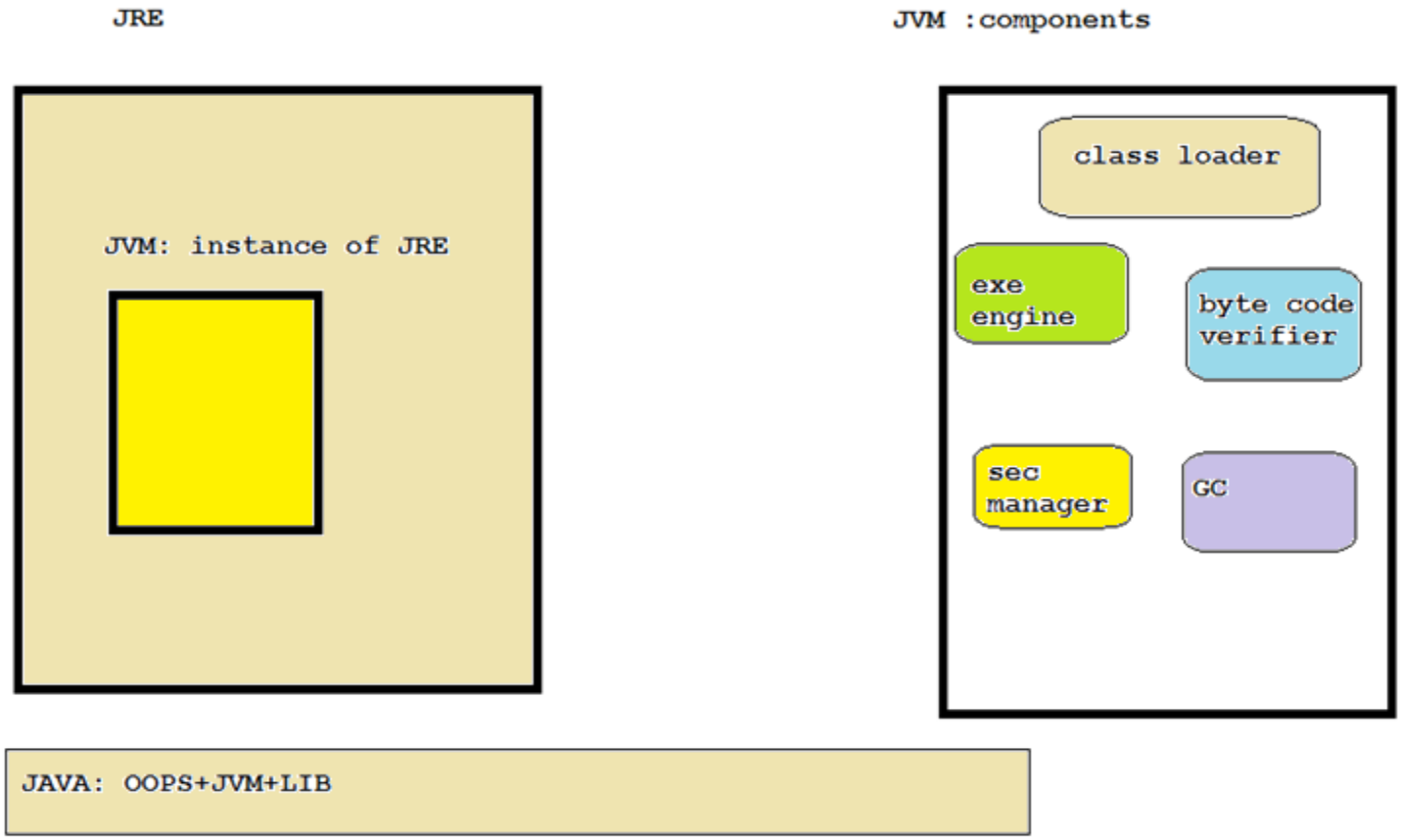
Called Java2

Version for the session

Some common buzzwords

- JDK
  - Java development kit, developer need it
- Byte Code
  - Highly optimize instruction set, designed to run on JVM
  - Platform independent
- JRE
  - Java Runtime environment, Create an instance JVM, must be there on deployment machine.
  - Platform dependent
- JVM
  - Java Virtual Machine JVM
  - Pick byte code and execute on client machine
  - Platform dependent…

Understanding JVM

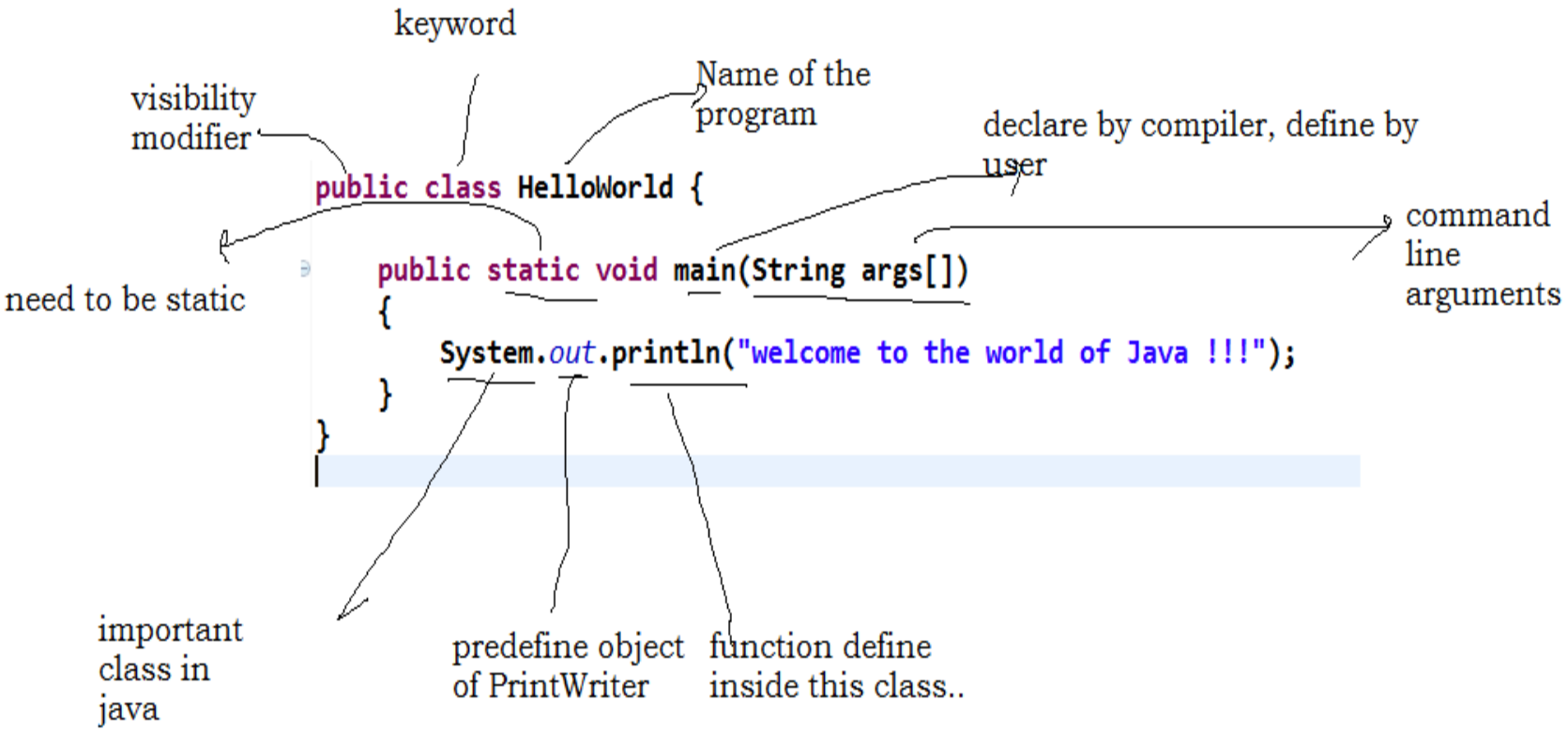


Hello World

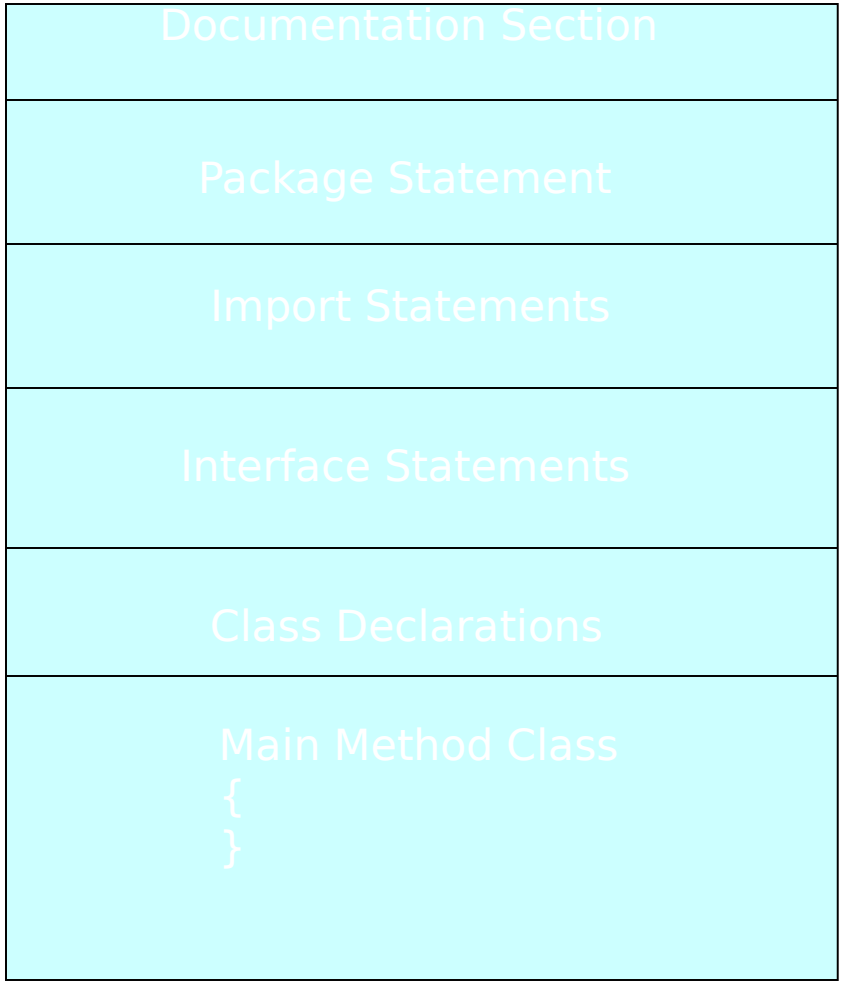
- Lab set-up
  - Java 1.7
  - Eclipse 3.6 (eclipse helios)

```
public class HelloWorld {

    public static void main(String args[])
    {
        System.out.println("welcome to the world of Java !!!");
    }
}
```

Analysis of Hello World !!!



Java Program Structure



Java Data Type

- 2 type
  - Primitive data type
  - Reference data type

Primitive data type

- boolean — either true of false
- char — 16 bit Unicode 1.1
- byte — 8-bit integer (signed)
- short — 16-bit integer (signed)
- int — 32-bit integer (signed)
- long — 64-bit integer (singed)
- float — 32-bit floating point (IEEE 754-1985)
- double — 64-bit floating point (IEEE 754-1985)

Java uses Unicode to represent characters internally

Procedural Programming

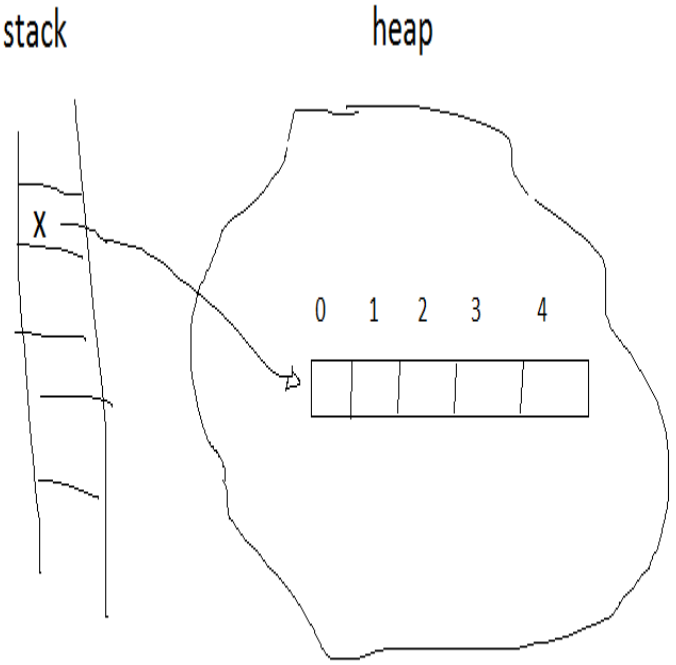
- if..else
- switch
- Looping; for, while, do..while as usual in Java as in C/C++

- Don’t mug the program/logic
- Follow dry run apprc
  - Try some programms
    - Create
    - Factorial program
    - Prime No check
    - Date calculation

Array

- One Dimensional ar
  - int x[]=new int[5]

How Java array differen C/C++ array?



Java Array: its different then C/C++

Session-2

- Object Oriented Concepts
- Object, class
- Basic principles of OO
  - Encapsulation
  - Abstraction
  - modularity
  - Inheritance/ Hierarchy
  - Polymorphism, message passing

rgupta.mtech@gmail.com

Object technologies

- OO is a way of looking at a software system as a collection of interactive objects

Reality

Tom's House

Tom

Tom's Car

Model

What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software.
  - Physical entity

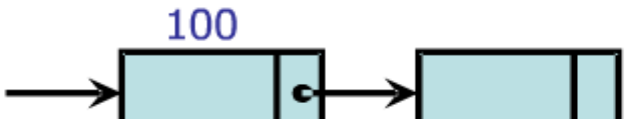
Tom's Car

  - Conceptual entity

US$100,000,000,000

Bill Gate's bank account

  - Software entity



rgupta.mtech@gmail.com

Class

- A class is the blueprint from which individual objects are created.
- An object is an instance of a class.

Object factory

Cookie Cutter

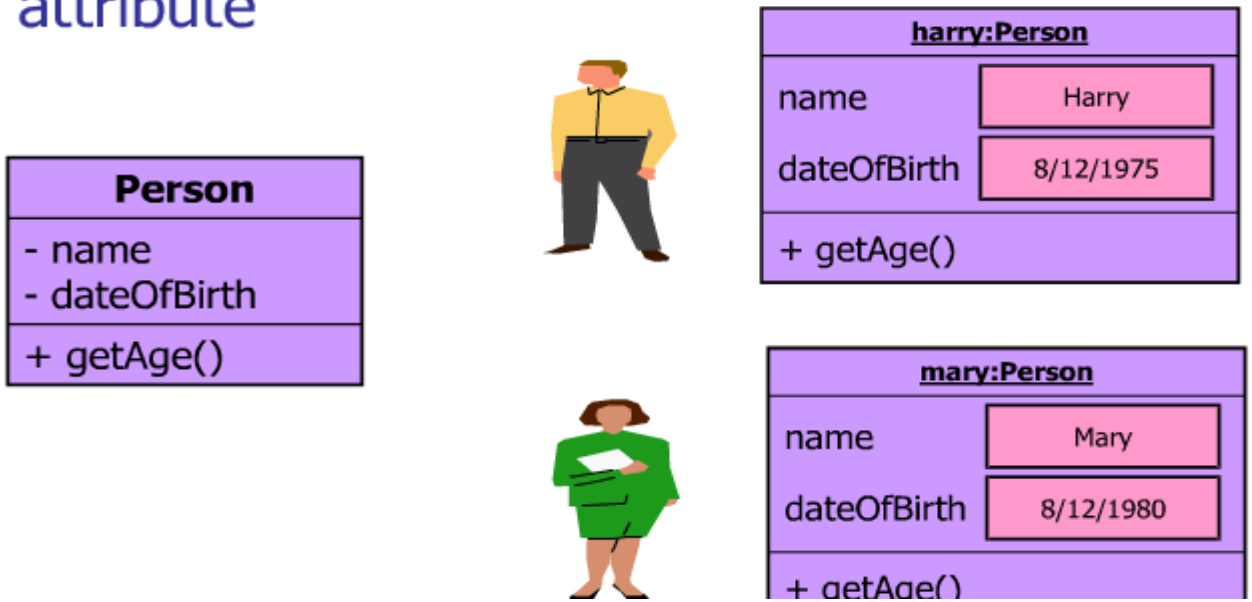
```
public class StudentTest {
  public static void main(String[] args) {
    Student s1 = new Student();
    Student s2 = new Student();
  }
}
```

```
- class -

public class Student {
  private String name;
  // …
}
```

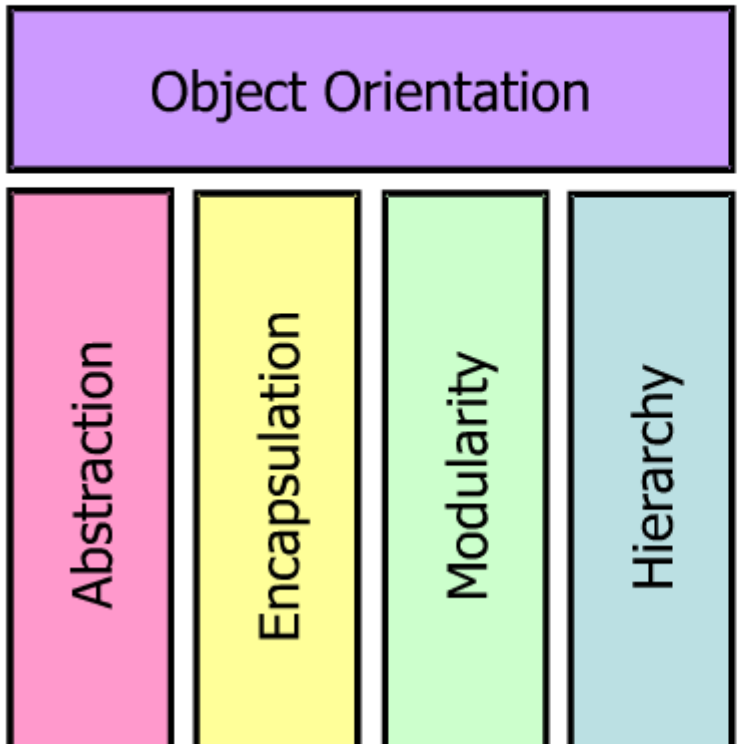
rgupta.mtech@gmail.com

Classes and Object

- All the objects share the same attribute names and methods with other objects of the same class
- Each object has its own value for each of the attribute

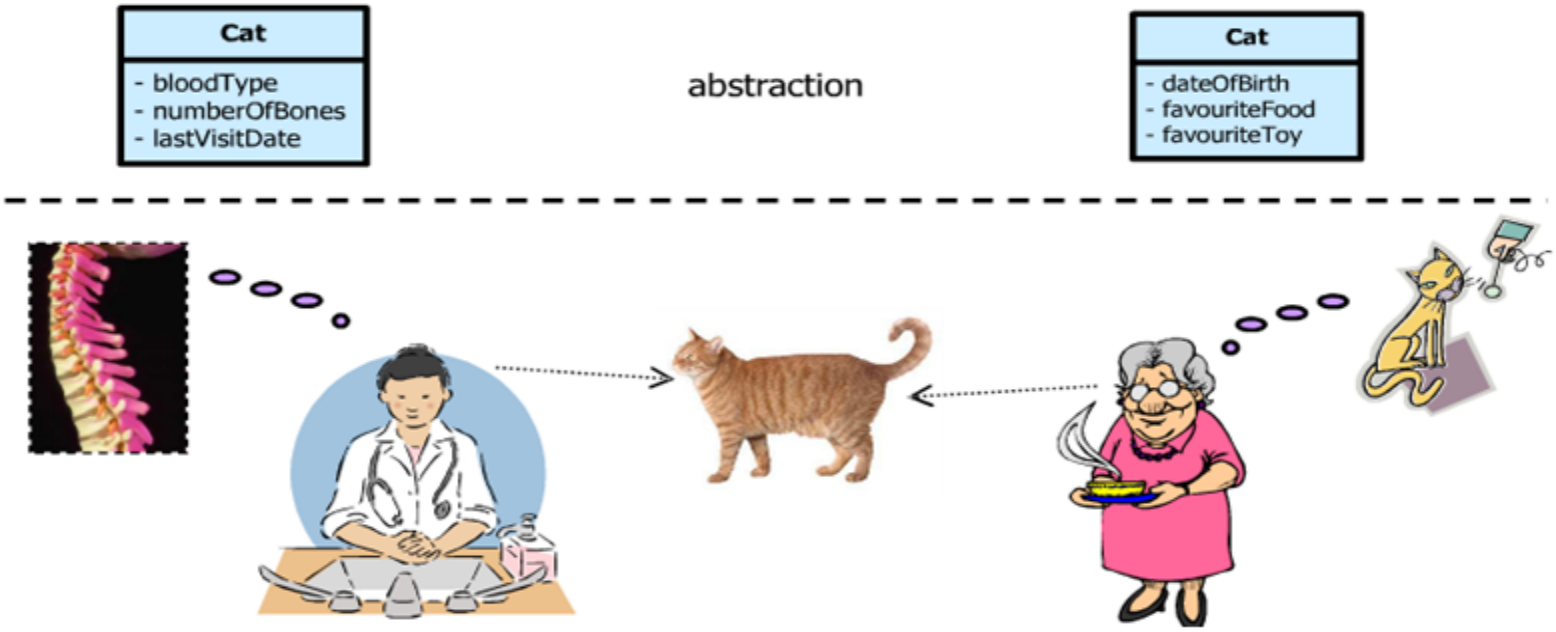


Basic principles of OO



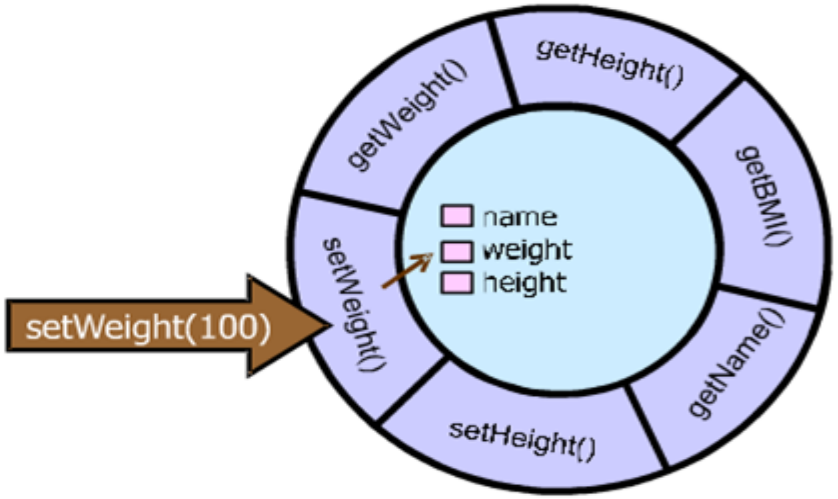
Abstraction

- Determine the relevant properties and features while ignoring non-essential details



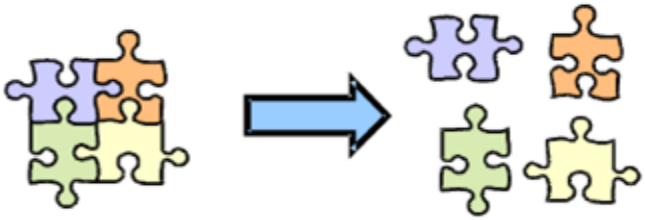
Encapsulation

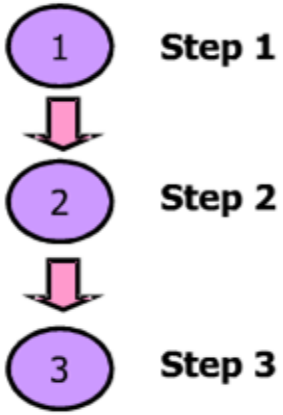
- Separate components into external and internal aspects



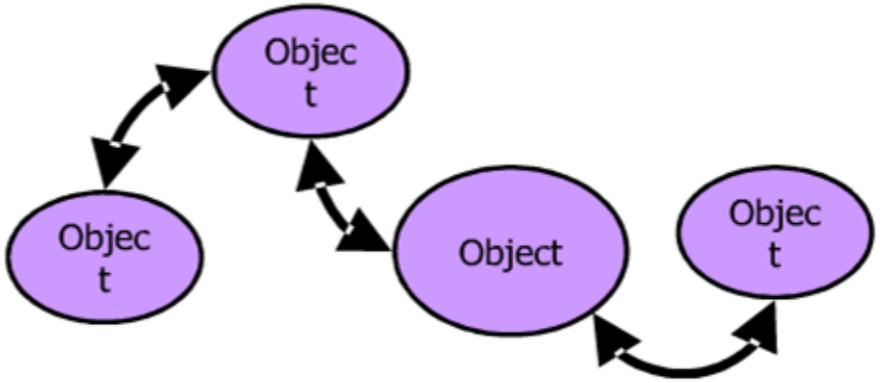
Modularity

- Break something complex into manageable pieces
  - Functional Decomposition
  - Object Decomposition





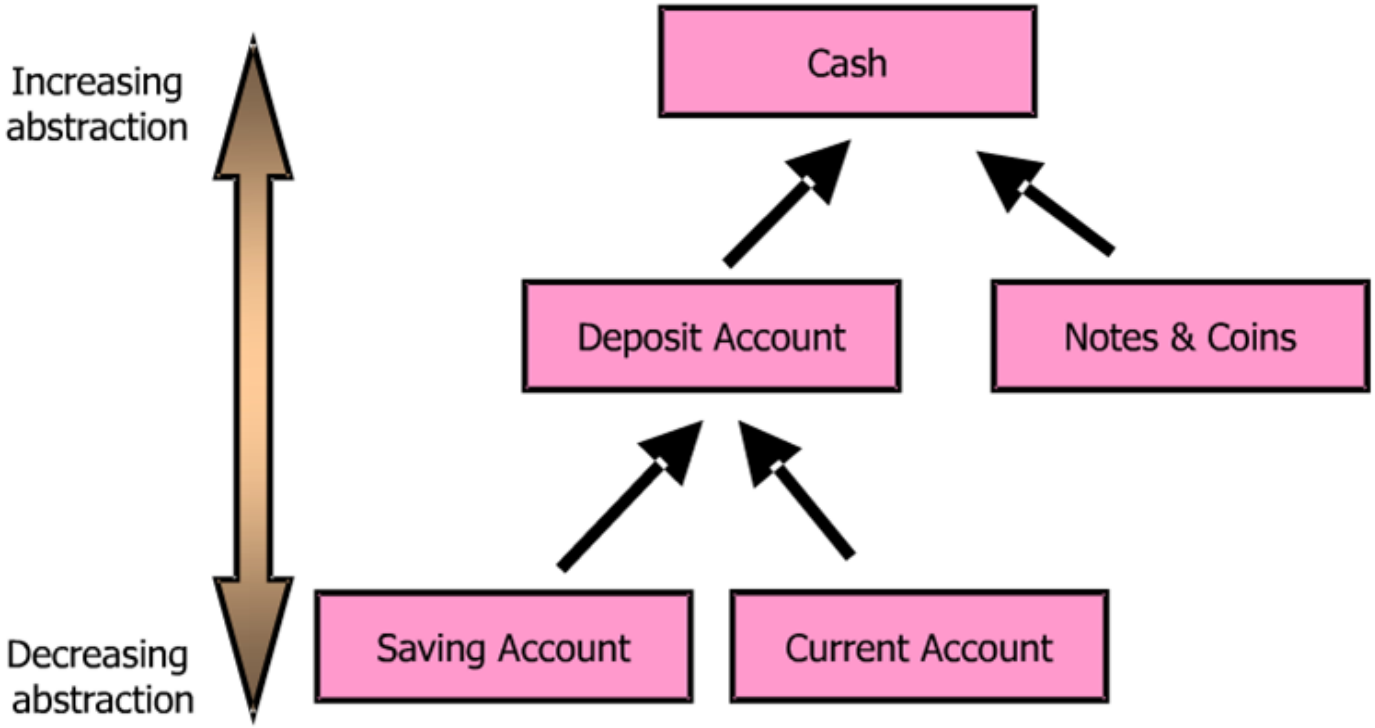
Functional Decomposition



Object Decomposition

Hierarchy

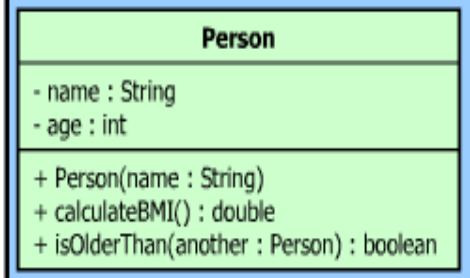
- Ranking or ordering of objects



A bit about UML diagram...

- UML 2.0 aka modeling language has 12 type of diagrams
- Most important once are class diagram, use case diagram and sequence diagram.
- You should know how to read it correctly
- This is not UML session... ☐

UML	Implementation

	`public class Person {`
	`  private String name;`
	`  private int age;`
	
	`  public Person(String name) { ...}`
	
	`  public double calculateBMI() { ... }`
	
	`  public boolean isOlderThan(Person another) {...}`
	`}`

DAY-2

rgupta.mtech@gmail.com

Session-1

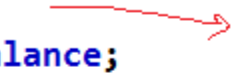
- Class and objects
  - Creating Classes and object
  - Object: state, identity and behaviour
  - Constructors: default, parameterized and copy
  - Need of “this” , Constructor chaining
  - Instance variable, static variable and local variable
  - Initialization block
  - Scanner and printf
  - Parameter passing in Java
    - Call by value / call by reference…

What can goes inside an class?

```
public class A {

    int i;          //  instance variable

    static int j;   // static variable

    //method in class
    public void foo(){
        int i;      //local variable
    }

    public A(){}            //default constructor

    public A(int j)         //parameterized ctr
    {
        //........
    }

    //getter and setter
    public int getI(){return i;}
    public void setI(int i){this.i=i;}
}
```

Creating Classes and object

```
class Account{

    public int id;
    public double balance;

    //.......
    //......

}

public class AccountDemo{
    public static void main(String[] args) {
        Account ac=new Account();
        ac.id=22;
    }
}
```



killing encapsulation

rgupta.mtech@gmail.com

Correct way?

```
class Account{
    private int id;
    private double balance;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public double getBalance() {
        return balance;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
}

public class AccountDemo{
    public static void main(String[] args) {
        Account ac=new Account();
        //ac.id=22; will not work
        ac.setBalance(2000);//correct way
    }
}
```

Constructors: default, parameterized and copy

- Initialize state of the object
- Special method have same name as that of class
- Can't return anything
- Can only be called once for an object
- Can be private
- Can't be static*
- Can overloaded but cant overridden*
- Three type of constructors
  - Default,
  - Parameterized and
  - Copy constructor

```
class Account{
    private int id;
    private double balance;

    //default ctr
    public Account() {
        //........
    }

    //parameterized ctr
    public Account(int i, double b) {
        this.id=i;
        this.balance=b;
    }

    //copy ctr
    public Account(Account ac) {
            //........
        }
}
```

Need of “this” ?

```
class Account{
   private int id;
   private double balance;

   //default ctr
   public Account() {
       //........
   }

   //parameterized ctr
   public Account(int id, double balance) {
       id=id;
       balance=balance;
   }

   //copy ctr
   public Account(Account ac) {
           //........
       }
```

which id assigned to which id ?

- Which id assigned to which id?
- “this” is an reference to the current object required to differentiate local variables with instance variables
  - Refer next slide…

“this” used to resolve confusion…

```
class Account{

    private int id;
    private double balance;

    //default ctr
    public Account() {
        //........
    }

    //parameterized ctr
    public Account(int id, double balance) {
        this.id=id;
        this.balance=balance;
    }

    //copy ctr
    public Account(Account ac) {
```

refer to instance variable

this : Constructor chaining?

- Calling one constructor from another ?

```
class Account{

    private int id;
    private double balance;

    //default ctr
    public Account() {
        this(22,555.0);
    }

    //parameterized ctr
    public Account(int id, double balance) {
        this.id=id;
        this.balance=balance;
    }

    //copy ctr
    public Account(Account ac) {
            //
```

Static method/variable

- Instance variable –per object while static variable are per class
- Initialize and define before any objects
- Most suitable for counter for object
- Static method can only access static data of the class
- For calling static method we don’t need an object of that class

How to count number of account object in the memory?

rgupta.mtech@gmail.com

Now guess why main was

Using static data..

```
class Account{

    private int id;
    private double balance;

    // will count no of account in application
    private static int totalAccountCounter=0;

    public Account(){
        totalAccountCounter++;
    }

    public  static int  getTotalAccountCounter(){
        return totalAccountCounter;
    }

}
```

static variable

static method

We can not access instance variable in static method but can access static variable in instance method

```
Account ac1=new Account();
Account ac2=new Account();

//How maany account are there in application ?

System.out.println(Account.getTotalAccountCounter());

System.out.println(ac1.getTotalAccountCounter());
```

Initialization block

- We can put repeated constructor code in an Initialization block...
- Static Initialization block runs before any constructor and runs only once...

```
class Account{

    private int id;
    private double balance;

    public Account(){
        //this is common code
    }
    public Account(int id , double balance){
        //this is common code

        this.id=id;
        this.balance=balance;
    }

}
```

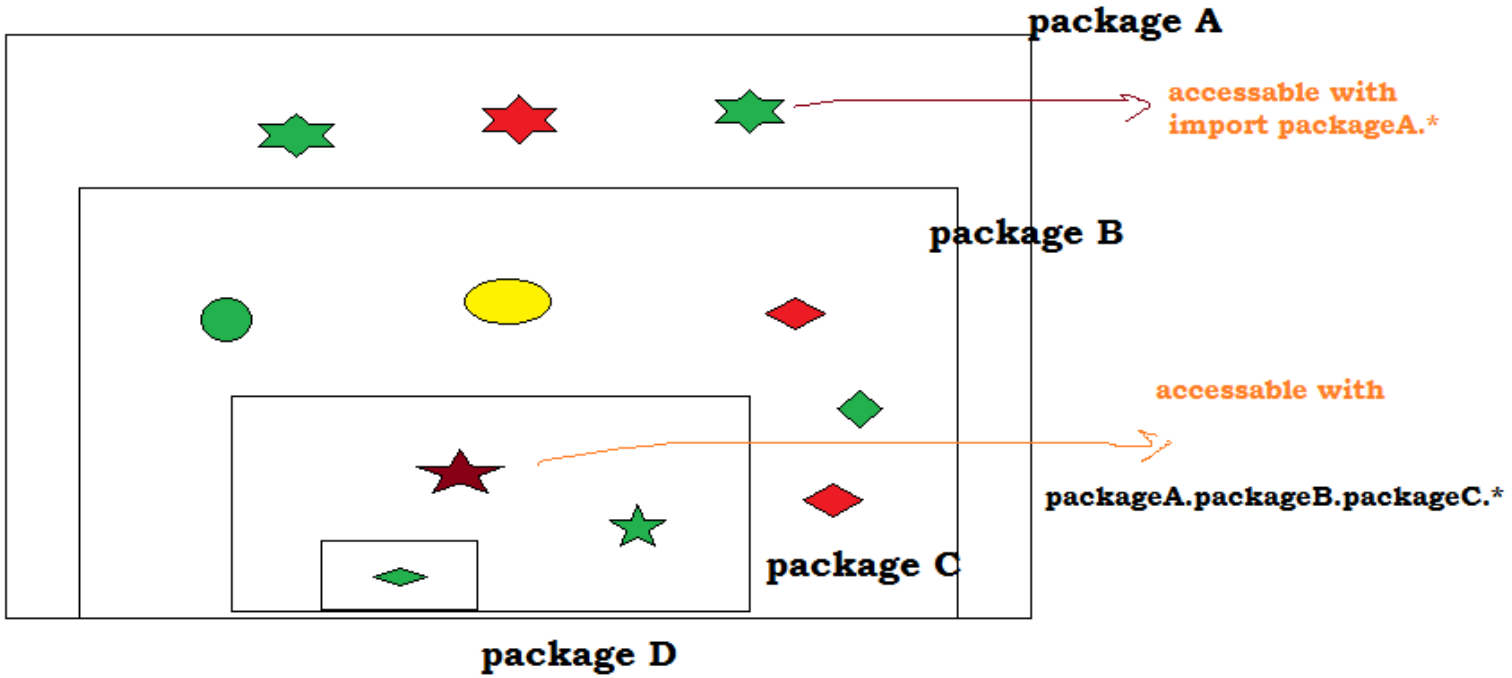
code repetition.

Initialization block

```
class Account{
   private int id;
   private double balance;
   private int accountCounter=0;

   static{
       System.out.println("static block: runs only once ...");
   }

   {
       System.out.println("Init block 1: this runs before any constructor  ...");
   }

   {
       System.out.println("Init block 2: this runs after inti  block 1 , before any const execute  ...");
   }
}
```

Packages

- Packages are Java's way of grouping a number of related classes and/or interfaces together into a single unit.
- Packages act as "containers" for classes.



Java Foundation Packages

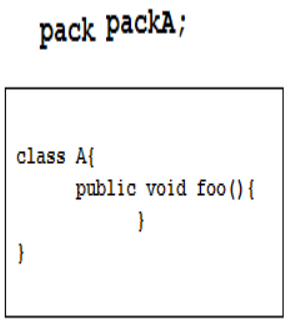
- Java provides a large number of classes groped into different packages based on their functionality.
- The six foundation Java packages are:
  - java.lang
    - Contains classes for primitive types, strings, math functions, threads, and exception
  - java.util
    - Contains classes such as vectors, hash tables, date etc.
  - java.io
    - Stream classes for I/O
  - java.awt
    - Classes for implementing GUI – windows, buttons, menus etc.
  - java.net
    - Classes for networking
  - java.applet
    - Classes for creating and implementing applets

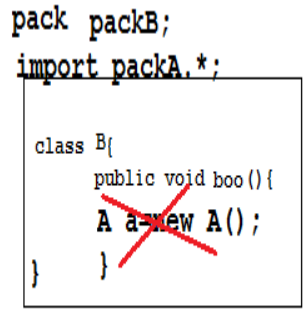
Visibility Modifiers

- For instance variable and methods
  - Public
  - Protected
  - Default (package level)
  - Private
- For classes
  - Public and default

Visibility Modifiers

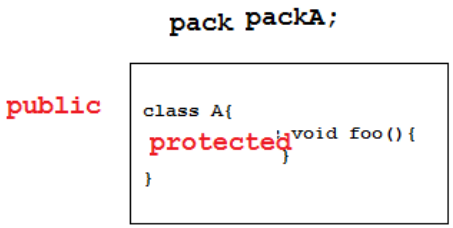
- class A has default visibility hence can access in the same package only.

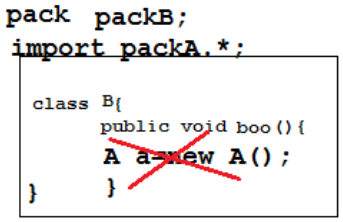


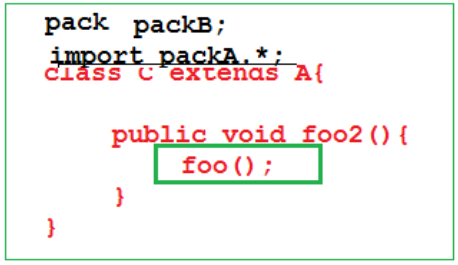


- Make class A public, then access it.

- Protected data can access in the same package and all the subclasses in other packages provide class itsef is public







Visibility Modifiers

rgupta.mtech@gmail.com

Want to accept parameter from user?

java.util.Scanner (Java 1.5)

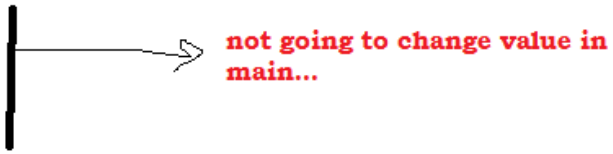
```
Scanner stdin = Scanner.create(System.in);

int n = stdin.nextInt();
String s = stdin.next();


boolean b = stdin.hasNextInt()
```

rgupta.mtech@gmail.com

Call by value

- Java support call by value
- The value changes in function is not going to reflected in the main.

```
public class CallByValue {
    public static void main(String[] args) {
        int i=22;
        int j=33;
        System.out.println("value of i before swapping:"+i);
        System.out.println("value of j before swapping:"+j);
        swap(i,j);
    }

     static void swap(int i, int j) {
        int temp;
        temp=i;
        i=j;
        j=temp;
    }
}
```



Call by reference

- Java don’t support call by reference.
- When you pass an object in an method copy of reference is passed so that we can mutate the state of the object but can’t delete original object itself

```
class Foo{
    private int i;
    public Foo(int i){
        this.i=i;
    }
    public int getI(){return i;}
    public void setI(int t){i=t;}
}
public class CallByref {

    public static void main(String[] args) {
        Foo f1=new Foo(22);
        Foo f2=new Foo(33);
        swap(f1,f2);
    }

     static void swap(Foo f1, Foo f2) {
        Foo temp;
        temp=f1;
        f1=f2;
        f2=temp;
        // f1.setI(55);
    }
```

do not effect f1 , f2 in main

can change state of f1

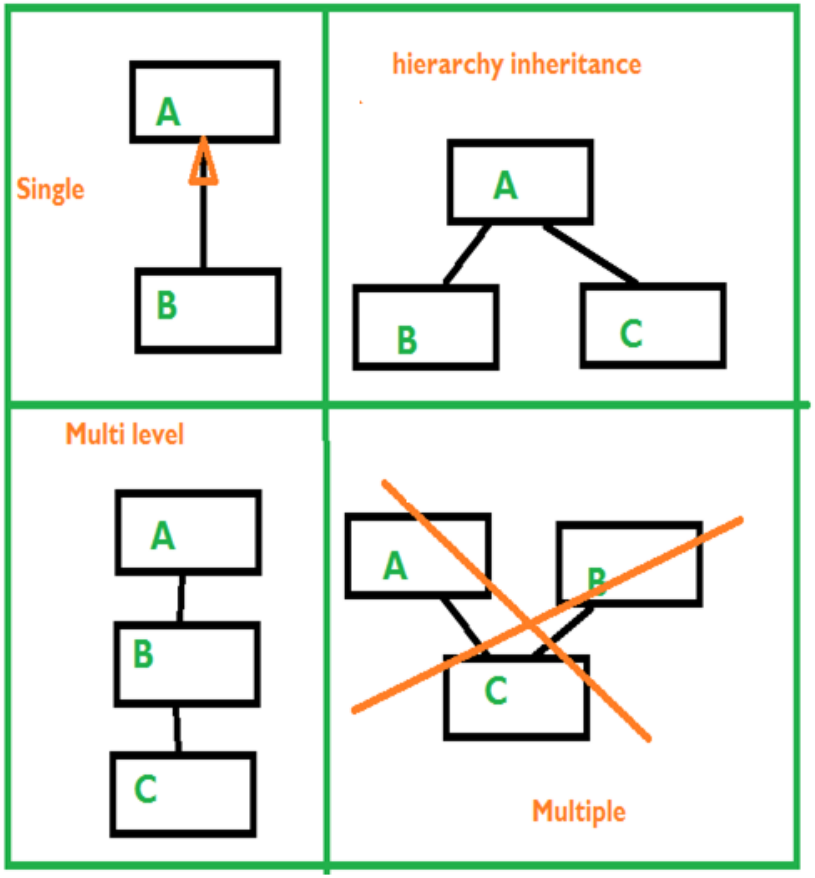
Session-2

- Inheritance
- Type of inheritance, diamond problem
- InstanceOf operator
- Final
- Final variable, final method , final class
- Acess control: public, private,protected and default
- Packages
- Abstract class
- Interface
- Polymorphism
- Overloading
- Overriding

rgupta.mtech@gmail.com

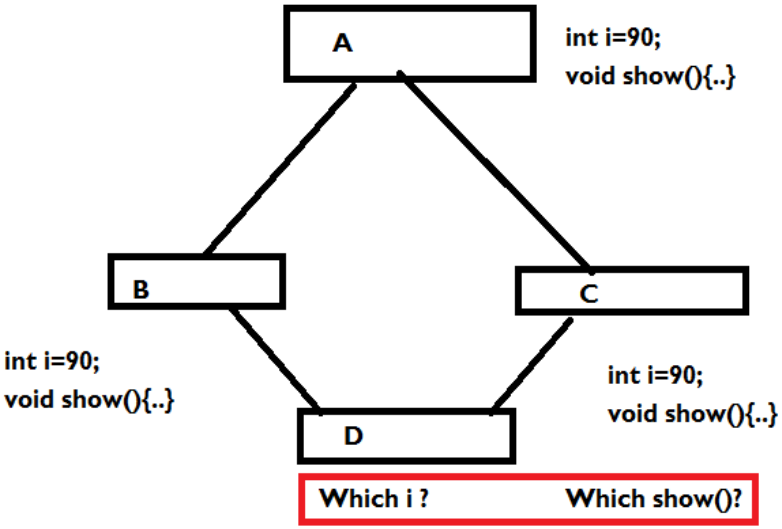
Inheritance

- ***Inheritance is the inclusion of behaviour (i.e. methods) and state (i.e. variables) of a base class in a derived class so*** that they are accessible in that derived class.
- **code reusability.**
- **Subclass and Super class concept**



Diamond Problem?

- Hierarchy inheritance can leads to poor design..
- Java don’t support it directly…(Possible using interface)



Inheritance example

- Use extends keyword
- Use super to pass values to base class constructor.

```
class A{
    int i;
    A()   {System.out.println("Default ctr of A");}

    A(int i) {System.out.println("Parameterized ctr of A");}
}

class B extends A{

    int j;
    B()   {System.out.println("Default ctr of B");}

    B(int i,int j)
    {
        super(i);
        System.out.println("Parameterized ctr of B");
    }

}
```

Overloading

- Overloading deals with multiple methods in the same class with the same name but different method signatures.

```
class MyClass {
    public void getInvestAmount(int rate) {…}

    public void getInvestAmount(int rate, long principal)
    { … }
}
```

Overriding…

- Overriding deals with two methods, one in the parent class and the other one in the child class and has the same name and signatures.

- Both the above methods have the same method names and the signatures but the method in the subclass *MyClass* overrides the method in the superclass *BaseClass*

- Overriding lets you define the **same operation in different ways for different object types.**

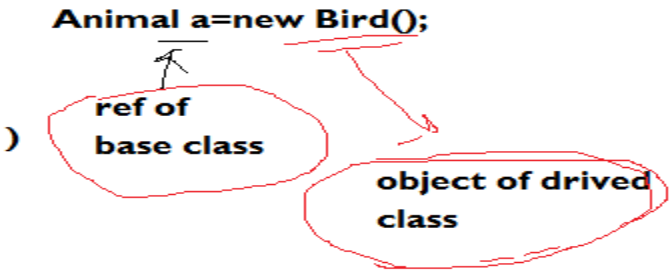
```
class BaseClass{
    public void getInvestAmount(int rate) {…}
}

class MyClass extends BaseClass {
    public void getInvestAmount(int rate) { …}
}
```

Polymorphism

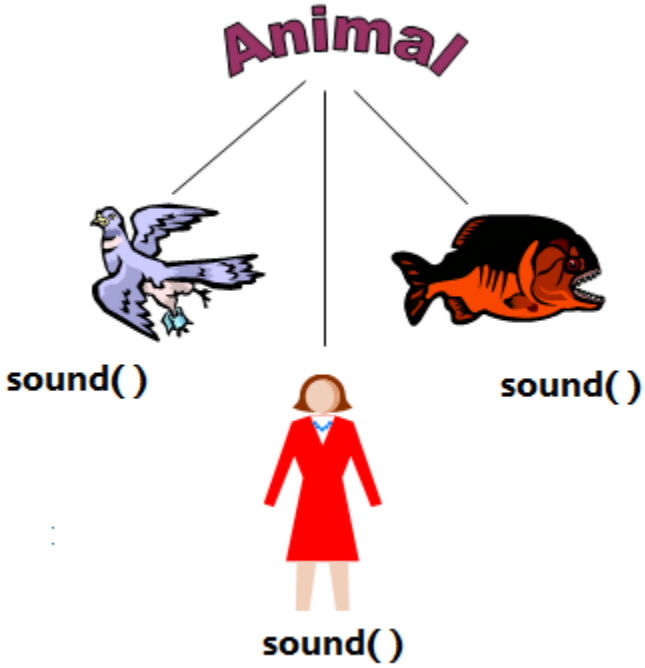
- Polymorphism=many forms of one things
- ***Substitutability***
- ***Overriding***
- ***Polymorphism means the ability of a single variable of a given type to be used to reference objects of*** different types, and automatically call the method that is specific to the type of object the variable references.

Polymorphism

- Every animal sound but differently…
- We want to have Pointer of Animal class assigned by object of derived class



Which method is going is to be called is not decided by the type of pointer rather object assigned will decide at run time



Example...

```
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

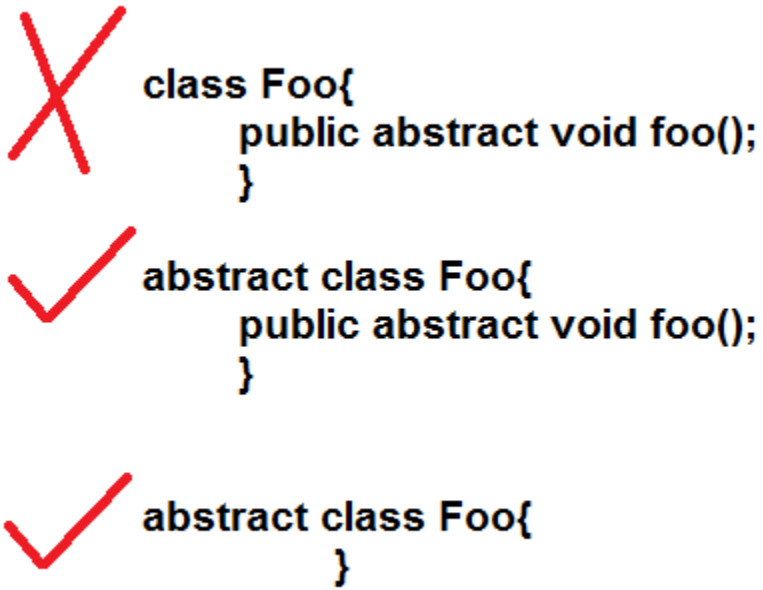
Need of abstract class?

- Sound( ) method of Animal class don't make any sense …i.e. it don't have semantically valid definition
- Method sound( ) in Animal class should be abstract means incomplete
- Using abstract method Derivatives of Animal class forced to provide meaningful sound() method

```
class Animal
{
    public void sound()
    {
        System.out.println("Don't know how generic animal sound.....");
    }
}
class Bird extends Animal
{@Override
    public void sound()
    {
        System.out.println("Bird sound.....");
    }
}
class Person extends Animal
{@Override
    public void sound()
    {
        System.out.println("Person sound.....");
    }
}
```

Abstract class

- If an class have at least one abstract method it should be declare abstract class.
- Some time if we want to stop a programmer to create object of some class...
- Class has some default functionality that can be used as it is.
- Can extends only one abstract class



Abstract class use cases…

- Want to have some default functionality from base class and class have some abstract functionality that can't be define at that mom

```
abstract class Animal
{
        public abstract void sound();
        public void eat()
         {
                  System.out.println("animal eat...");
         }
}
```

- Don't want to allow a programmer to create object of an class as it is too generic

More example...

```
public abstract class Account  {
   public void deposit (double amount) {
        System.out.println("depositing " + amount);
   }

   public void withdraw (double amount) {
        System.out.println ("withdrawing " + amount);
   }

   public abstract double calculateInterest(double  amount);
}
```

```
public class SavingsAccount extends Account {

    public double calculateInterest (double amount) {
       // calculate interest for SavingsAccount
       return amount * 0.03;
    }

    public void deposit (double amount) {
       super.deposit (amount); // get code reuse
       // do something else
    }

    public void withdraw (double amount) {
       super.withdraw (amount); // get code reuse
       // do something else
    }
}
```

```
public class TermDepositAccount extends Account {

   public double calculateInterest (double amount) {
      // calculate interest for SavingsAccount
      return amount * 0.05;
   }

   public void deposit(double amount) {
      super.deposit (amount); // get code reuse
      // do something else
   }

   public void withdraw(double amount) {
      super.withdraw (amount); // get code reuse
      // do something else
   }
}
```

Final

- What is the meaning of final
  - Something that can not be change!!!
- final
  - Final method arguments
    - Cant be change inside the method
  - Final variable
    - Become constant, once assigned then cant be changed
  - Final method
    - Cant overridden
  - Final class
    - Can not inherited (Killing extendibility)
    - Can be reuse

rguptamtech@gmail.com

Some examples

Final class

- Final class can't be subclass i.e. Can't be extended
- No method of this class can be overridden
- Ex: String class in Java…
- Real question is in what situation somebody should declare a class final

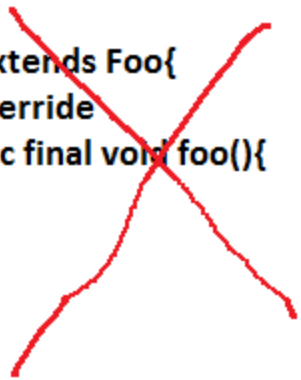
```
package cart;

public final class Beverage{

     public void importantMethod(){
          Sysout("hi");
     }
}
```

```
package examStuff;
import cart.*;

class Tea extends beverage{

}
```

Final Method

- Final Method Can’t overridden
- Class containing final method can be extended

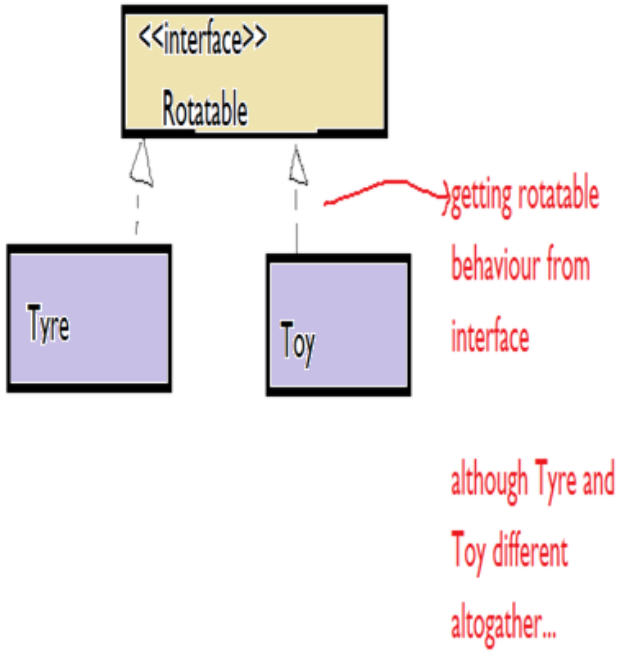
```
class Foo{

        public final void foo(){
                Sysout("i am the best");
                Sysout("You can use me but can't override me");
        }
};

class Bar extends Foo{
        @Override
        public final void foo(){


        }
}
```



Interface?

- Interface : Contract bw two parties
- Interface method
  - Only declaration
  - No method definition
- Interface variable
  - Public static and final constant
    - Its how java support global constar
- Break the hierarchy
- Solve diamond problem
- Callback in Java*

Some Example ….



Interface?

- Rules
  - All interface methods are always public and abstract, whether we say it or not.
  - Variable declared inside interface are always public static and final
  - Interface method can’t be static or final
  - Interface cant have constructor
  - An interface can extends other interface
  - Can be used polymorphically
  - An class implementing an interface must implement all of its method otherwise it need to declare itself as an abstract class…

Implementing an interface...

What we declare

```
interface Bouncable{
     int i=9;
    void bounce();
    void setBounceFactor();
}
```

What compiler think...

```
interface Bouncable{
    public static final int i=9;
    public abstract void bounce();
    public abstract void setBounceFactor();
}
```

All interface method must be implemented....

```
class Tyre implements Bouncable{

    public void bounce(){
        Sysout(i);
        Sysout(i++);
    }
    public void setBounceFactor(){}
}
```

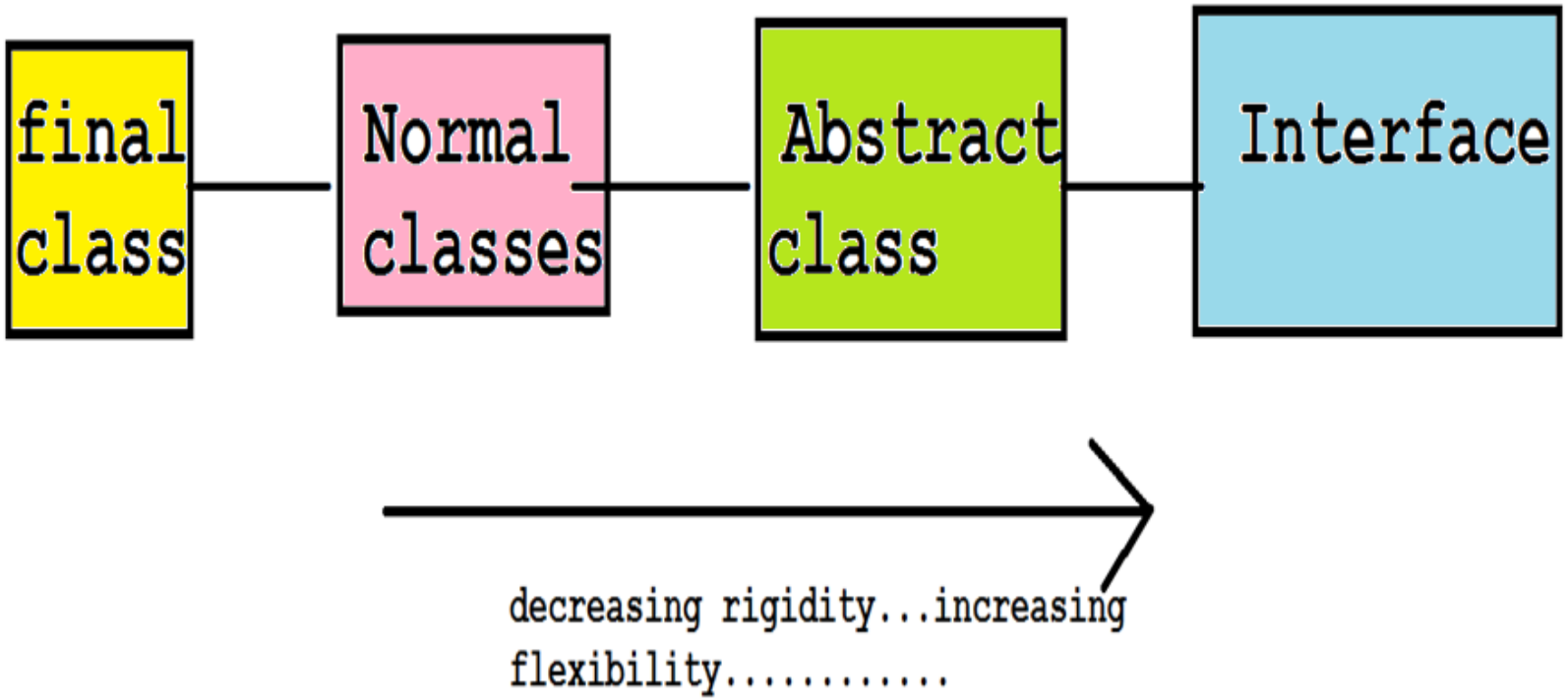
rgupta.mtech@gmail.com

Note

- Following interface constant declaration are identical
  - int i=90;
  - public static int i=90;
  - public int i=90;
  - public static int i=90;
  - public static final int i=90;

- Following interface method declaration don’t compile
  - final void bounce();
  - static void bounce();
  - private void bounce();
  - protected void bounce();

Decreasing Rigidity..increasing Flexibility



Type of relationship bw objects

- USE-A
- HAS-A
- IS-A (Most costly ?)

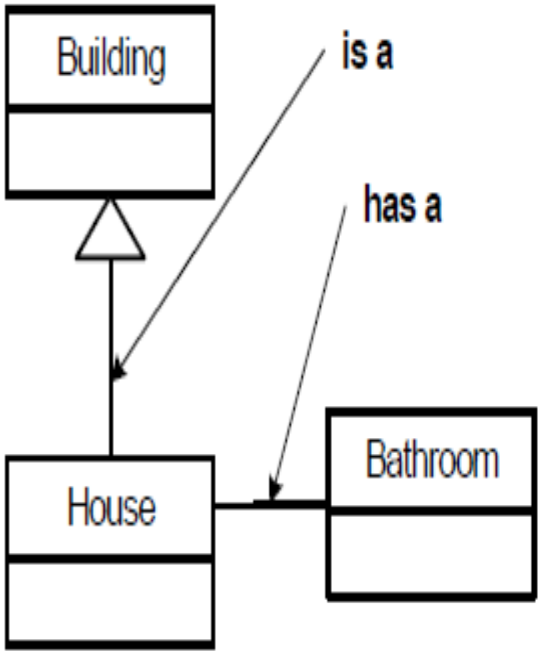
ALWAYS GO FOR LOOSE COUPLING AND HIGH COHESION...

But HOW?

rgupta.mtech@gmail.com

IS-A VS HAS-A

Inheritance [is a] Vs Composition [has a]



is a [House is a Building]

```
class Building{
 .....
}

class House extends Building{
 .....
}
```

has a [House has a Bathroom]

```
class House {
    Bathroom room = new Bathroom() ;
    ....
   public void getTotMirrors(){
       room.getNoMirrors();
       ....
   }
}
```

rgupta.mtech@gmail.com

DAY-3

rgupta.mtech@gmail.com

Session-1

- String class
  - Immutability and usages,
  - stringbuilder and stringbuffer
- Wrapper classes and usages
- Java 5 Language Features (I):
  - AutoBoxing and Unboxing
  - Enhanced For loop
  - Varargs
  - Static Import
  - Enums
- Inner classes
  - Regular inner class
  - method local inner class
  - anonymous inner class

rgupta.mtech@gmail.com

String

- Immutable i.e. once assigned then can't be changed
- Only class in java for which object can be created with or without using new operator

Ex: String s="india";
String s1=new String("india"); What is the difference?

- String concatenation can be in two ways:
  - String s1=s+ "paki"; Operator overloading
  - String s3=s1.concat("paki");

Immutability

- Immutability means something that cannot be changed.
- Strings are immutable in Java. What does this mean?
  - String literals are very heavily used in applications and they also occupy a lot of memory.
  - Therefore for efficient memory management, all the strings are created and kept by the JVM in a place called string pool (which is part of Method Area).
  - Garbage collector does not come into string pool.
  - How does this save memory?

Some common string methods...

- charAt()
  - Returns the character located at the specified index
- concat()
  - Appends one String to the end of another ("+" also works)
- equalsIgnoreCase()
  - Determines the equality of two Strings, ignoring case
- length()
  - Returns the number of characters in a String
- replace()
  - Replaces occurrences of a character with a new character
- substring()
  - Returns a part of a String
- toLowerCase()
  - Returns a String with uppercase characters converted
- toString()
  - Returns the value of a String
- toUpperCase()
  - Returns a String with lowercase characters converted
- trim()
  - Removes whitespace from the ends of a String

String comparison

- Two string should never be checked for equality using == operator
  WHY?
- Always use equals( ) method....

```
String s1="india";
String s2="paki";

    if(s1.equals(s2))
        .....
        .....
```

Wrapper classes

- Helps treating primitive data as an object
- But why we should convert primitive to objects?
  - We can't store primitive in java collections
  - Object have properties and methods
  - Have different behavior when pa
    argument
- Eight wrapper for eight primiti
  - Integer, Float, Double, Character

```
Integer it=new Integer(33);
int temp=it.intValue();
....
```

primitive==>object

```
Integer it=new Integer(i);
```

object==>primitive

```
int i=it.intValue()
```

primitive ==>string

```
String s=Integer.toString();
```

String==>Numeric object

```
Double val=Double.valueOf(str)
```

rgupta.mtech@gmail.com

Boxing / Unboxing Java 1.5

- Boxing

Integer iWrapper = 10;
Prior to J2SE 5.0, we use
Integer a = new Integer(10);

- Unboxing

int iPrimitive = iWrapper;
Prior to J2SE 5.0, we use
int b = iWrapper.intValue();

Java 5 Language Features (I)

- AutoBoxing and Unboxing
- Enhanced For loop
- Varargs
- Covariant Return Types
- Static Import
- Enums

rgupta.mtech@gmail.com

Enhanced For loop

- Provide easy looping construct to loop through array and collections

```
int x[]=new int[5];
....
....
for(int temp:x)
        Sysout("temp:"+temp);

class Animal{ }
class Cat extends Animal{   }
class Dog extends Animal{  }

Animal []aa={new Cat(),new Dog(), new Cat()};

for(Animal a:aa)
        .......
```

Varargs

- Java start supporting variable argument 1.5

```
class Foo{

        public void foo(int ...j)
        {
                for(int temp:j)
                        Sysout(temp);
        }
        .....
        .....
}
```

- Discuss function overloading in case of Varargs and wrapper classes

Static Import

- Handy feature in Java 1.5 that allow something like this:
- import static java.lang.Math.PI;

import static java.lang.Math.cos;

- Now rather then using
  - ***double r = Math.cos(Math.PI * theta);***
- We can use something like
  - ***double r = cos(PI * theta);*** – looks more readable ....

rgupta.mtech@gmail.com

- Avoid abusing static import like

Enums

- Enum is a special type of classes.
- enum type used to put restriction on the instance values

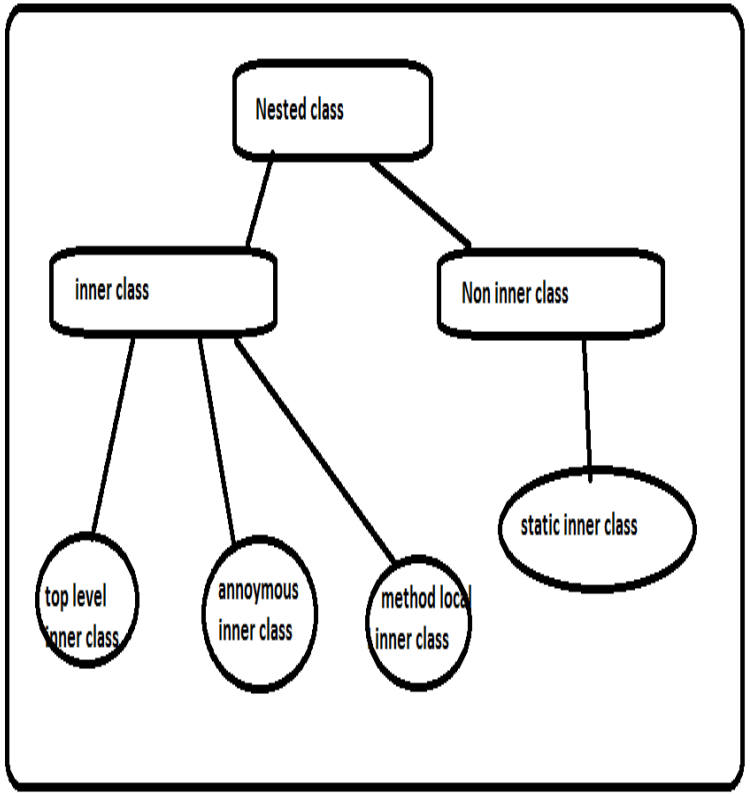
```
public enum myCars{
    HONDA,BMW,TOYOTA
};
```

its optional !!!

```
        ........
        ........
    myCars currentcar=myCars.BMW;
     System.out.println("My Current Cars : "+ myCars.BMW);
```

```
enum Day {
        SUNDAY, MONDAY, TUESDAY,WEDNESDAY, THURSDAY,
        FRIDAY, SATURDAY
    }

//used
Day today = Day.WEDNESDAY;
switch(today){
        case SUNDAY:
                break;
        //...
}
```

Inner classes



- Inner class?
  - A class defined inside another class.
  - Why to define inner classes
- Inner classes
  - Top level inner class
  - Method local inner class
  - Anonymous Inner class
  - Method local argument inner class
- Static inner classes

Top level inner class

- Non static inner class object can't be created without outer class instance
- All the private data of outer class is available to inner class
- Non static inner class can't have static members
  - http://www.avajava.com/tutorials/lessons/iterator-pattern.html

```
class A
{
    private int i=90;

    class B
    {
        int i=90;//
        void foo()
        {
            System.out.println("instance value of outer class:"+ A.this.i);

            System.out.println("instance value of inner class:"+this.i);
        }
    }
}
public class Inner1 {

    public static void main(String[] args) {

        A.B objectB=new A().new B();

        objectB.foo();
```

Method local inner class

- Class define inside an method
- Can not access local data define inside method
- Declare local data final to access it

```
class A
{
    private int i=90;
    void foo()
    {
        int i=22;
        final int j=44;
        class B
        {
            void foofy()
            {
                //System.out.println("value of method local variable cant be access:"+i);
                System.out.println("value of method local" +
                        " can be accessed if it is declared final:"+ j);
            }
        }

        B b=new B();
        b.foofy();
    }

}
```

rgupta.mtech@gmail.com

Anonymous Inner class

- A way to implement polymorphism
- “On the fly”

```
interface Cookable
{
    public void cook();
}

class Food
{
    Cookable c=new Cookable() {
        @Override
        public void cook() {
            System.out.println("Cooked......");
        }
    };
}
```

Session-2

- Exception handling
  - Try, catch, throw, throws, finally
  - Checked and Unchecked exception
  - User defined exceptions
  - Rule for Overloading/ Overriding in case of method throwing exceptions
  - Assertions

rgupta.mtech@gmail.com

What is Exception?

- An exception is an abnormal condition that arises while running a program.

Examples:

- Attempt to divide an integer by zero causes an exception to be thrown at run time.
- Attempt to call a method using a reference that is null.
- Attempting to open a nonexistent file for reading.
- JVM running out of memory.

- Exception handling do not correct abnormal condition rather it make our program robust i.e. make us enable to take remedial action when exception

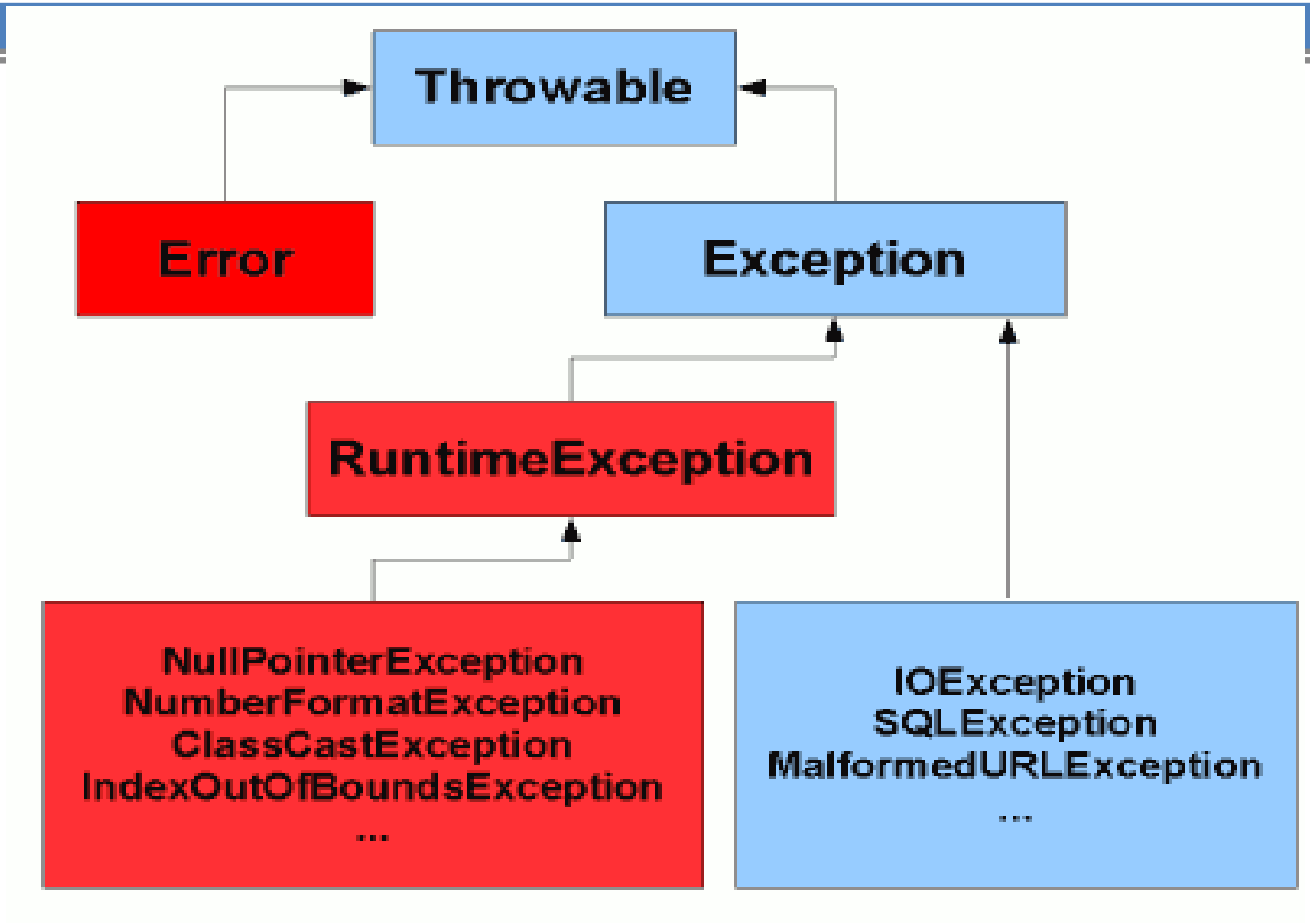
Type of exceptions

1. Unchecked Exception
   - Also called Runtime Exceptions
2. Checked Exception
   - Also called Compile time Exceptions
3. Error
   - Should not to be handled by programmer..like JVM crash

- All exceptions happens at run time in programming and also in real life.....
- for checked ex, we need to tell java we know about those problems for example readLine() throws IOException

- Exception key words
- Catch
- Throw

Exception Hierarchy



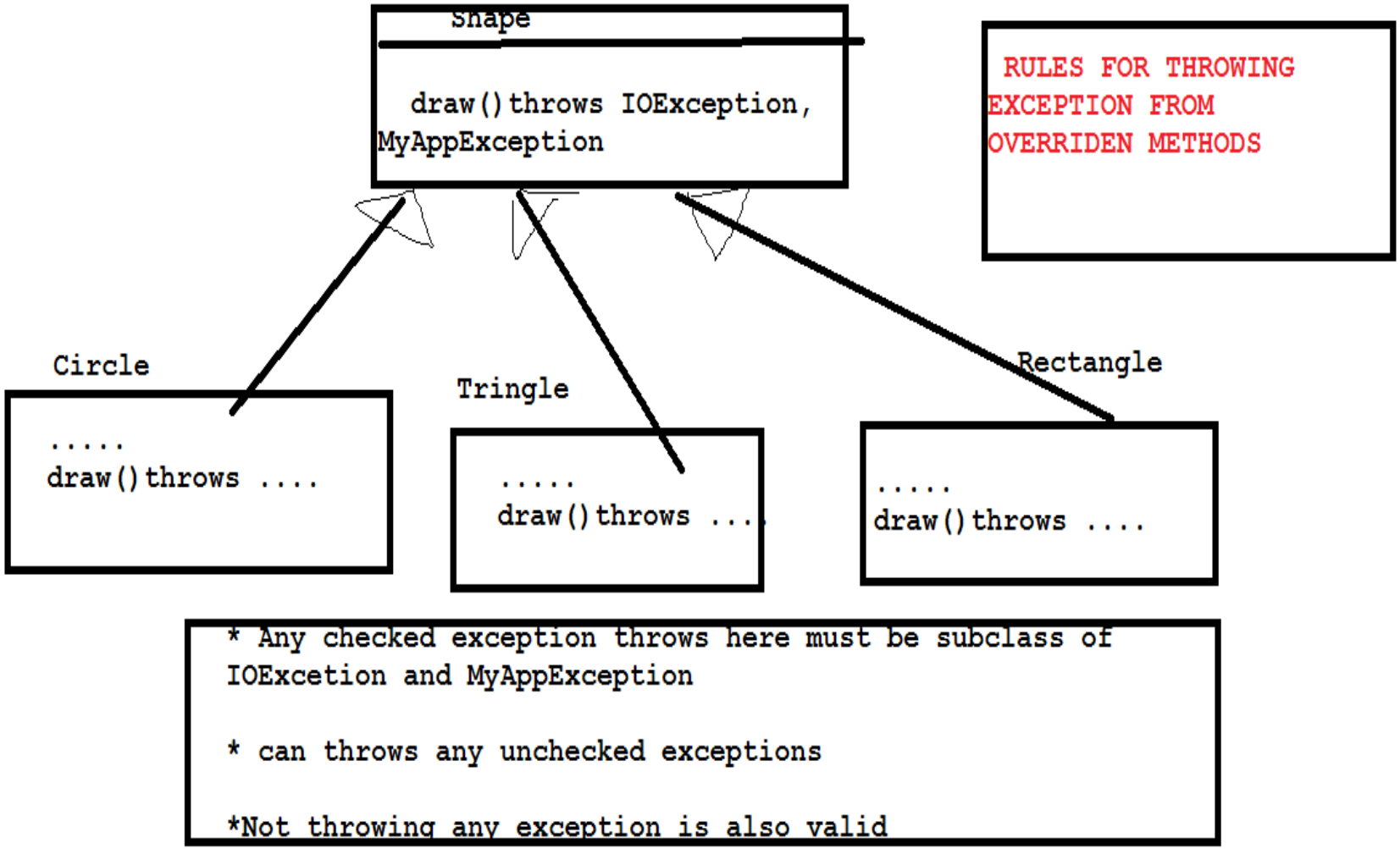
Exceptions and their Handling

- When the JVM encounters an error such as divide by zero, it creates an exception object and throws it – as a notification that an error has occurred.
- If the exception object is not caught and handled properly, the interpreter will display an error and terminate the program.
- If we want the program to continue with execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then take appropriate corrective actions. This task is known as *exception handling*.

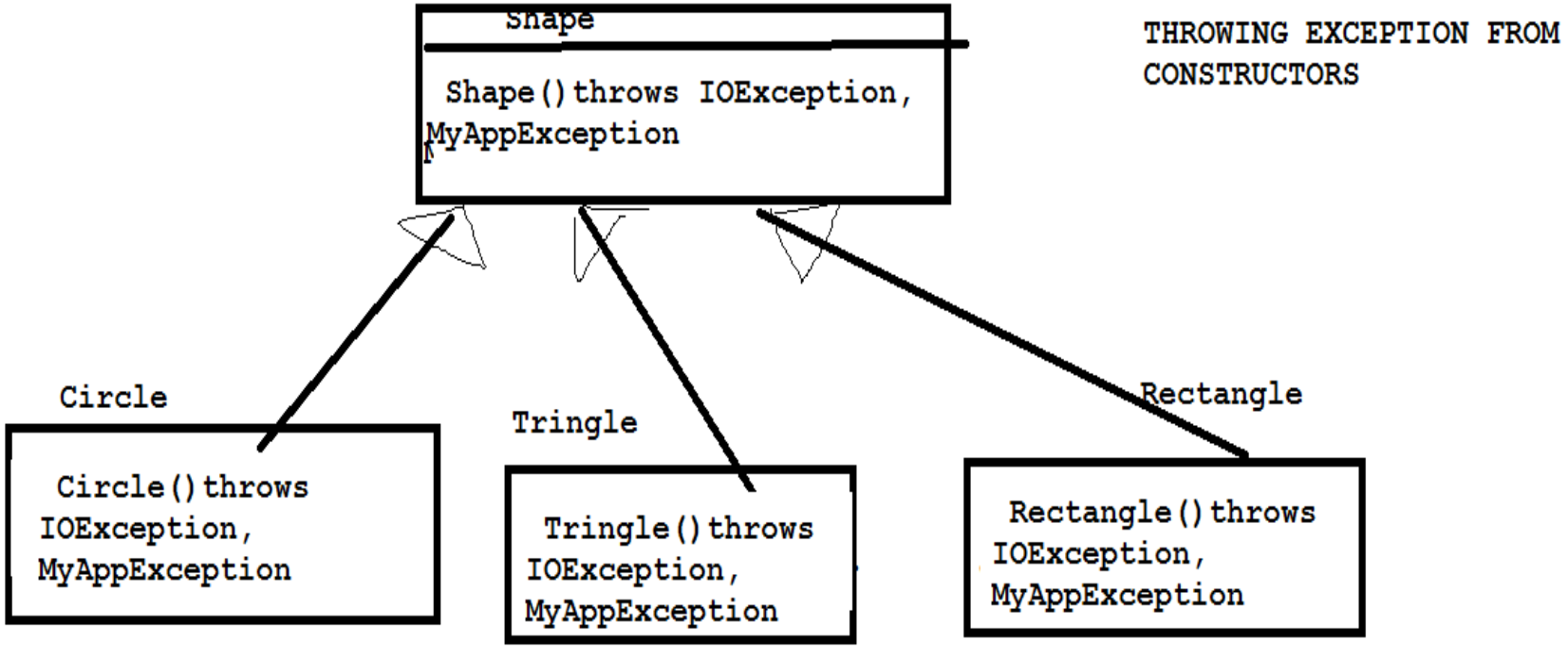
Common Java Exceptions

- ArithmeticException
- ArrayIndexOutOfBoundException
- ArrayStoreException
- FileNotFoundException
- IOException – general I/O failure
- NullPointerException – referencing a null object
- OutOfMemoryError
- SecurityException – when applet tries to perform an action not allowed by the browser's security setting.
- StackOverflowError
- StringIndexOutOfBoundException

Rules for throwing exception from overridden



Rules for throwing exception from constructors

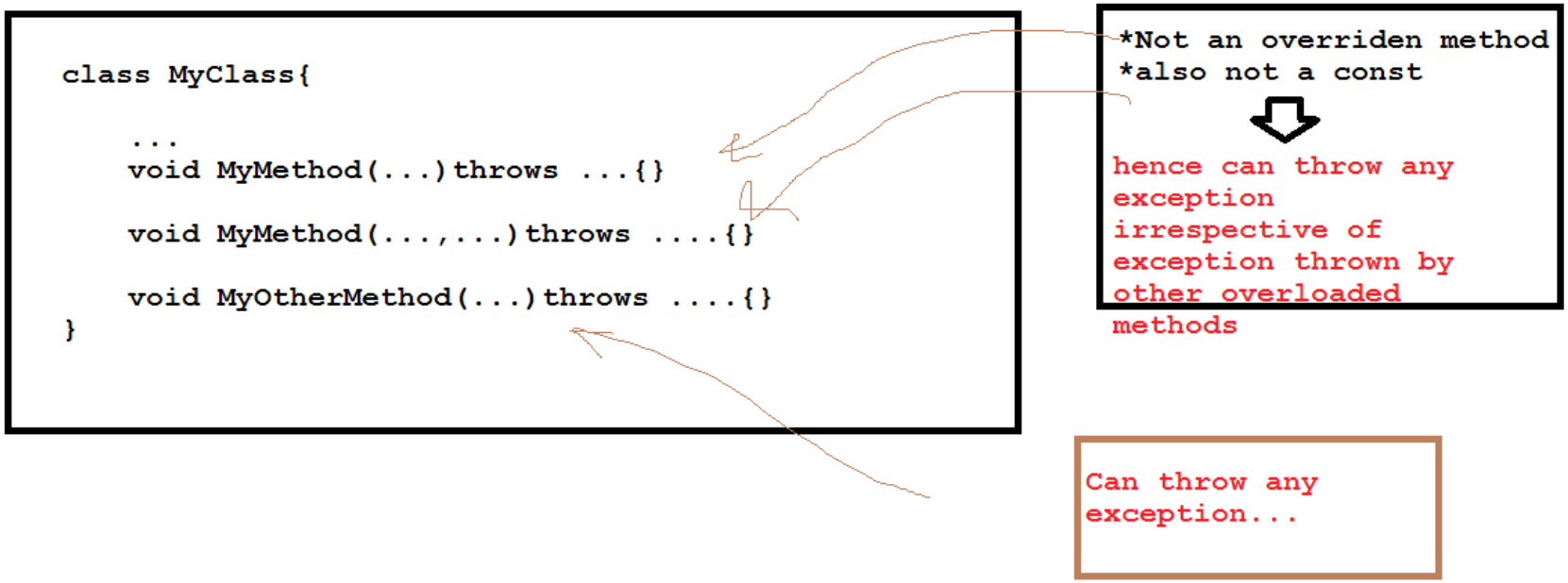


* Must throws IOException and MyAppException

* May throws additional checked exceptions

Rules for throwing exception from overloaded and other methods

Throwing Exception from Overloaded and other methods...



DAY-4

rgupta.mtech@gmail.com

Session-1

- Char and byte oriented streams
- BufferedReader and BufferedWriter
- File handling
- Object Serialization
  [ObjectInputStream / ObjectOutputStream]

rgupta.mtech@gmail.com

Stream

- Stream is an abstraction that either produces or consumes information.
- A stream is linked to a physical device by the Java I/O system.
- All streams behave in the same manner, even if the actual physical devices to which they are linked differ.
- 2 types of streams:
  - byte : for binray data
    - All byte stream class are like XXXXXXXXStream
  - character: for text data
    - All char stream class are like XXXXXXXXReader/ XXXXXXWriter

Some important classes form java.io

TABLE 6-1 java.io Mini API

java.io Class	Extends From	Key Constructor(s) Arguments	Key Methods
File	Object	File, String String String, String	createNewFile() delete() exists() isDirectory() isFile() list() mkdir() renameTo()
FileWriter	Writer	File String	close() flush() write()
BufferedWriter	Writer	Writer	close() flush() newLine() write()
PrintWriter	Writer	File (as of Java 5) String (as of Java 5) OutputStream Writer	close() flush() format()*, printf()* print(), println() write()
FileReader	Reader	File String	read()
BufferedReader	Reader	Reader	read() readLine()
			*Discussed later

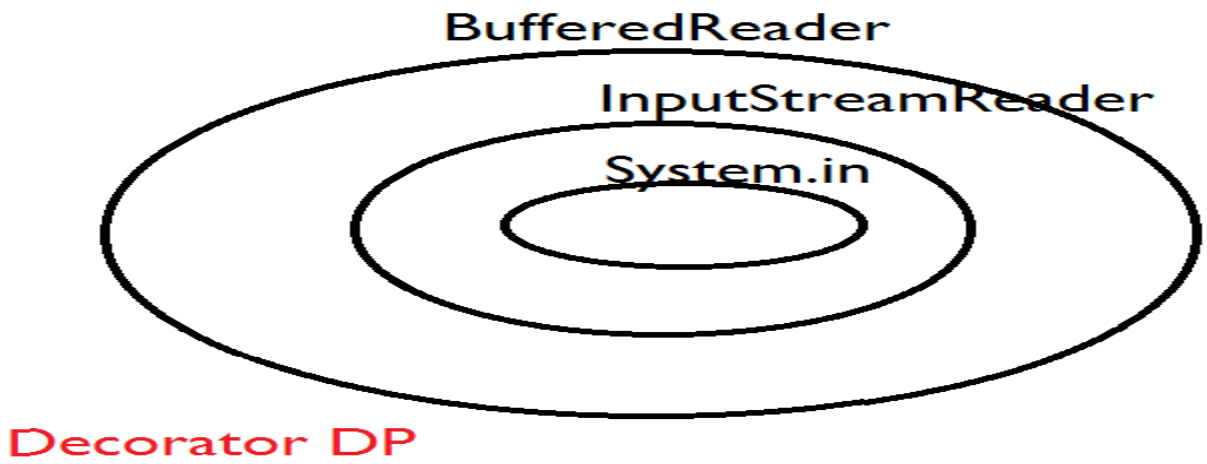
System class in java

- System class defined in java.lang package
- It encapsulate many aspect of JRE

- System class also contain 3 predefine stream variables
  - in
    - System.in (InputStream)
  - out
    - System.out(PrintStream)
  - err
    - System.err(console)

BufferedReader and BufferedWriter

- Reading form console
  - BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
- Reading form file
  - BufferedReader br=**new BufferedReader(new FileReader(new File("c:\\raj\\foo.txt")));**



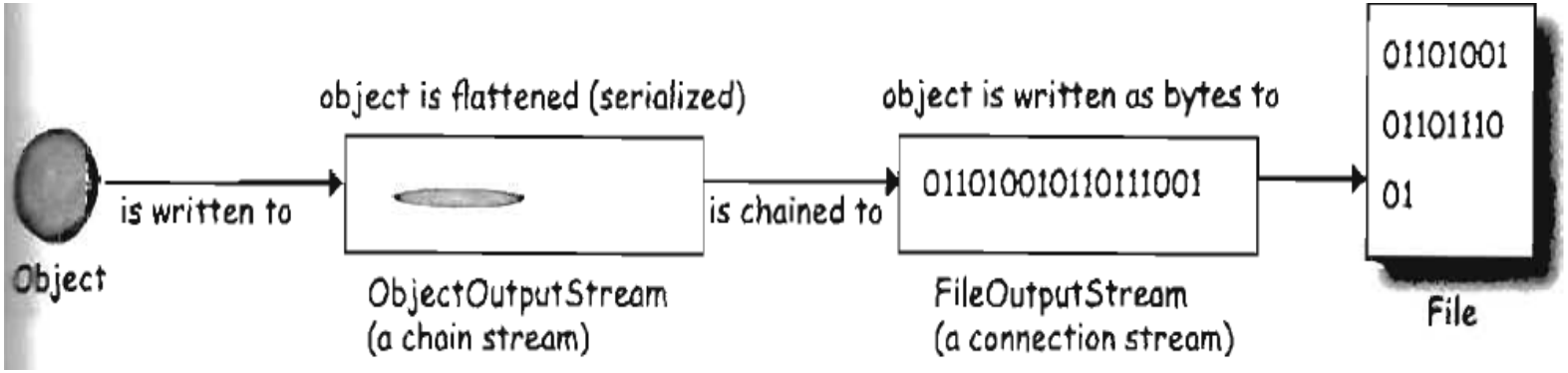
File

- File abstraction that represent file and directories
  - File f=new File("....");
  - boolean flag= file.createNewFile();
  - boolean flag= file.mkDir();
  - boolean flag=file.exists();
- FileWriter

```
File file = new File( "fileWrite2.txt");
FileWriter fw =new FileWriter(file);
fw.write("howdy\nfolks\n"); // write characters to
fw.flush(); // flush before closing
fw.close(); // close file when done
```

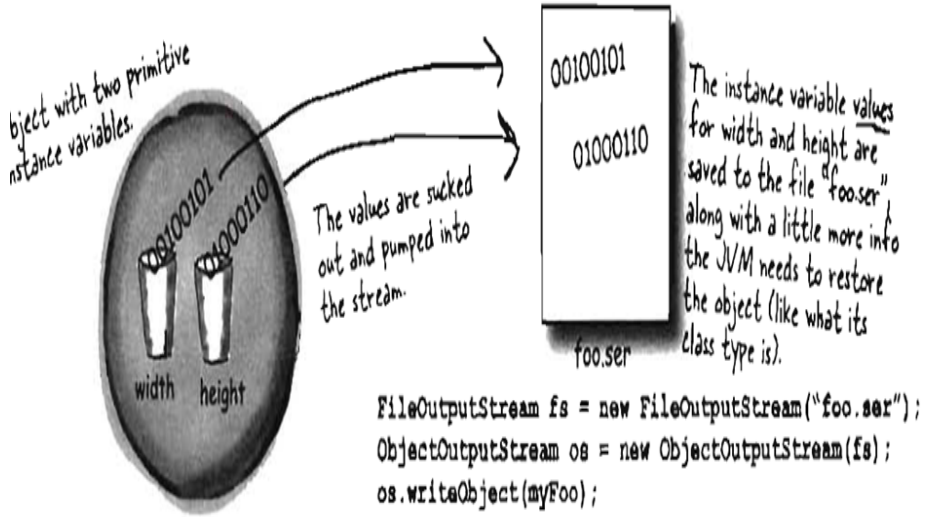
Serialization

- Storing the state of the object on a file along some metadata….so that it Can be recovered back……..
- Serialization used in RMI (Remote method invocation) while sending an object from one place to another in network…



What actually happens during Serialization

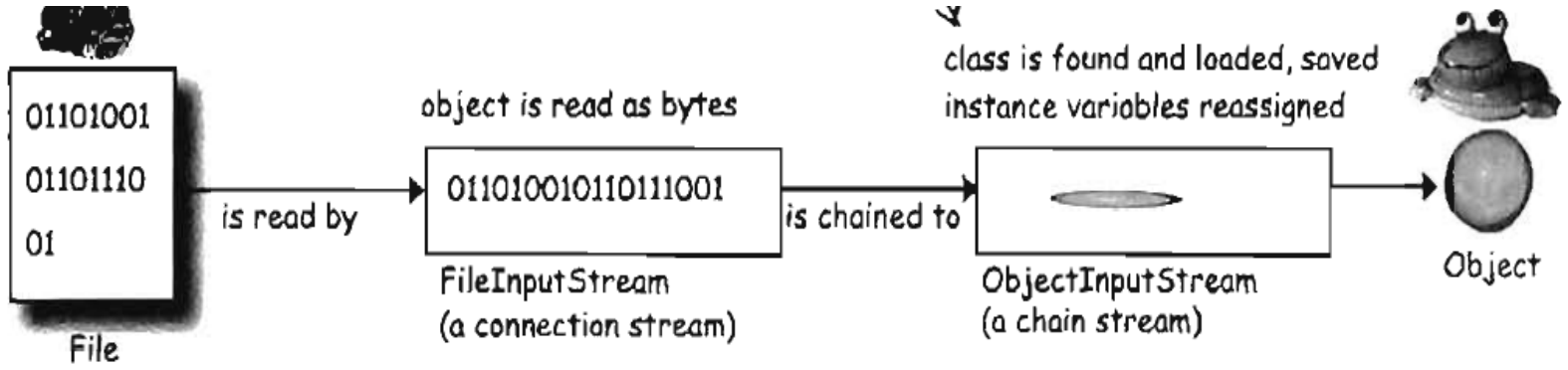
- The state of the object from heap is sucked and written along with some meta data in an file….



When an object is serialized, all the objects it refers to from instance variables are *also* serialized. And all the objects *those* objects refer to are serialized. And all the objects *those* objects refer to are serialized… and the best part is, it happens automatically!

De- Serialization

- When an object is de-serialized, the JVM attempts to bring object back to the life by making an new object on the heap that have the same state as original object
- Transient variable don’t get saved during serialization hence come with null !!!



Hello world Example...

```
class Box implements Serializable{
	private static final long serialVersionUID = 1L;
	int l,b;

	public Box(int l, int b) {
		super();
		this.l = l;
		this.b = b;
	}

	public String toString() {
		return "Box [l=" + l + ", b=" + b + "]";
	}

}
```

```
Box box=new Box(22, 33);
```

```
FileOutputStream fo=new FileOutputStream("c:\\raj\\foofoo.ser");
ObjectOutputStream os=new ObjectOutputStream(fo);

os.writeObject(box);
```

```
box=null;//nullify to prove that object come by de-ser...
```

```
FileInputStream fi=new FileInputStream("c:\\raj\\foofoo.ser");
ObjectInputStream oi=new ObjectInputStream(fi);

box=(Box)oi.readObject();
```

```
System.out.println(box);
```

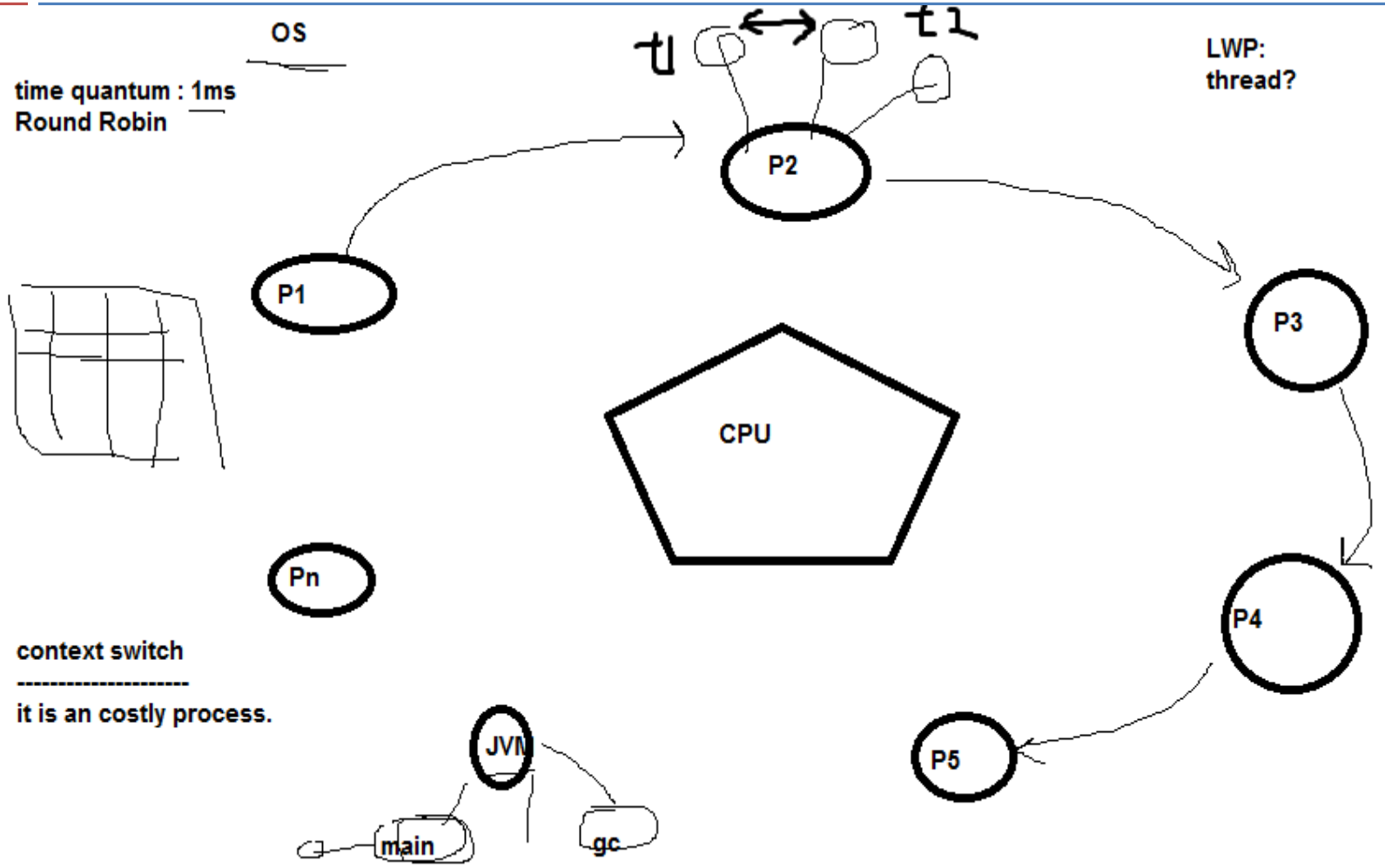
rgupta.mtech@gmail.com

Session-2

- Multi-Threading
  - Creating threads: using Thread class and Runnable interface
  - Thread life cycle
  - Using sleep(), join(), thread priorities
  - Synchronization
  - Solving producer and consumer problem using wait() and notify()
  - Deadlock introduction

rgupta.mtech@gmail.com

What is threads? LWP



rgupta.mtech@gmail.com

Basic fundamentals

- Thread: class in java.lang
- thread: separate thread of execution

What happens during multithreading?

1. JVM calls main() method
2. main() starts a new thread. Main thread is tempory frozen while new thread start running so JVM switch between created thread and main thread till both complets

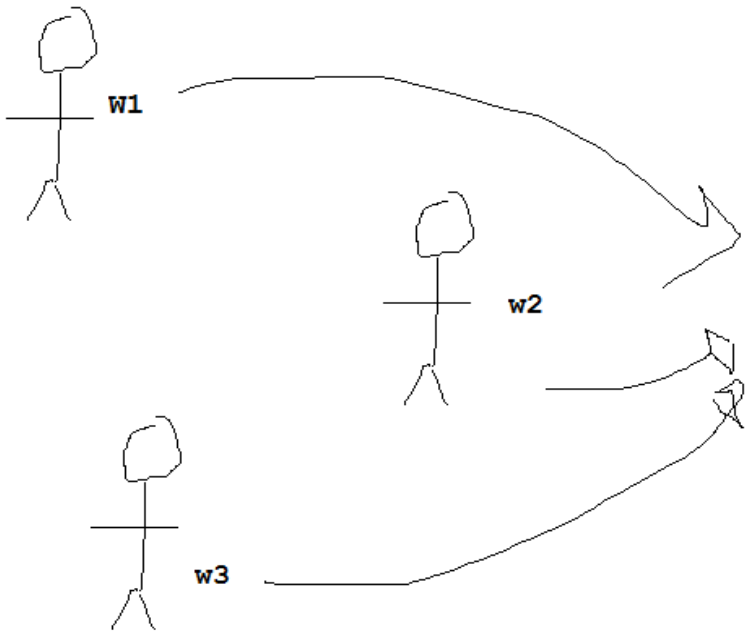
Creating threads Java?

- Implements Runnable interface
- Extending Thread class……..

- Job and Worker analogy…

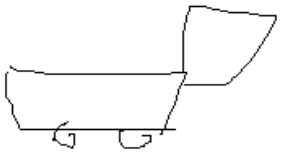
```
class Job implements Runnable{


    public void run() {
        // TODO Auto-generated method stub
    }

}
```

```
class MyThread extends Thread{
    public void run() {
        // TODO Auto-generated method stub
    }
}
```



Thread
----
consider object of threads as worker

Implementation of Runnable as Job



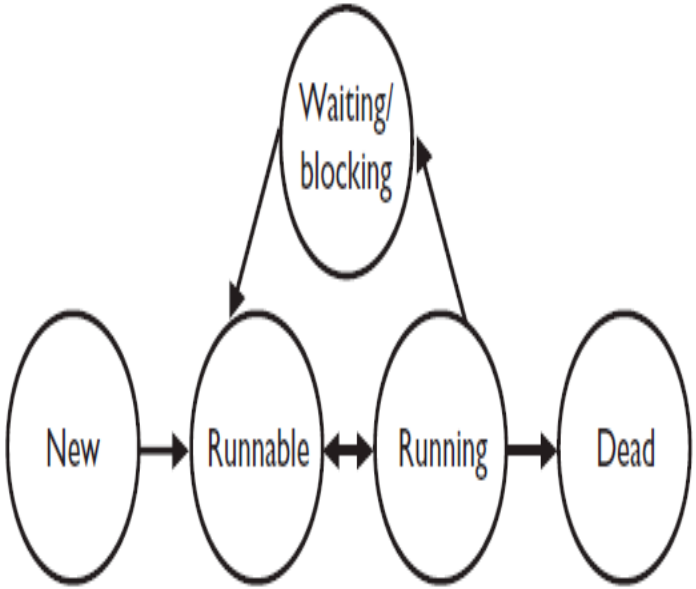
simulation

Sleep()

Job

Thread life cycle...

- If a thread is blocked (put to sleep) specified no of ms should expire
- If thread waiting for IO that must be over..
- If a threadcalls wait() then another thread must call notify() or notifyAll()
- If an thred is suspended() some one must call resume()



Java.lang.Thread

- Thread()
  - construct new thread
- void run()
  - must be overriden
- void start()
  - start thread call run method
- static void sleep(long ms)
  - put currently executing thread to sleep for specified no of millisecond
- boolean isAlive()
  - return true if thread is started but not expired
- void stop()
- void suspend() and void resume()
  - Suspend thread execution….

Java.lang.Thread

- void join(long ms)
  - Main thread Wait for specified thread to complete or till specified time is not over
- void join()
  - Main thread Wait for specified thread to complete
- static boolean interrupted()
- boolean isInterrupted()
- void interrupt()
  - Send interrupt request to a thread

Creating Threads

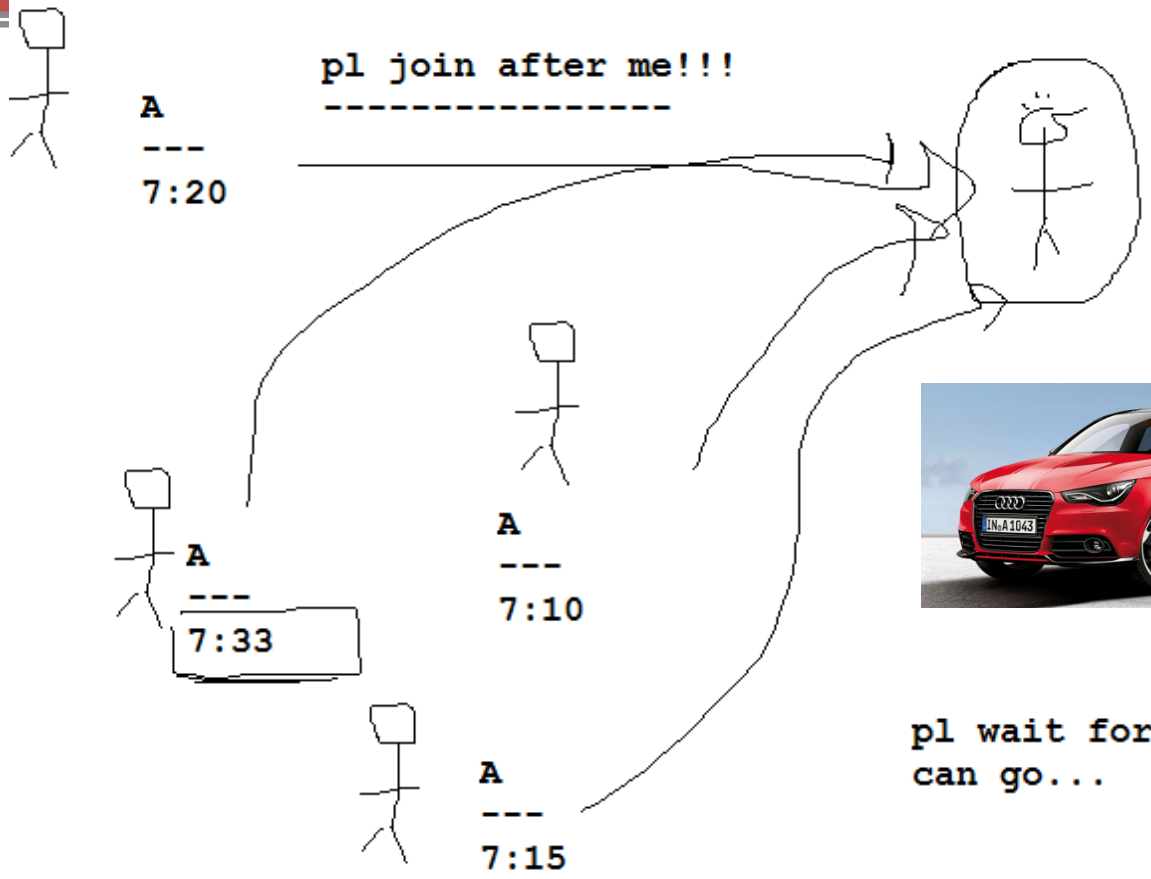
- By extending Thread class

```
class MyThread extends Thread
{
    @Override
    public void run()
    {

    }

}
```

- Implementing the Runnable interface

```
class MyRunnable implements Runnable
{

    @Override
    public void run()
    {

    }

}
```

Understanding join() method



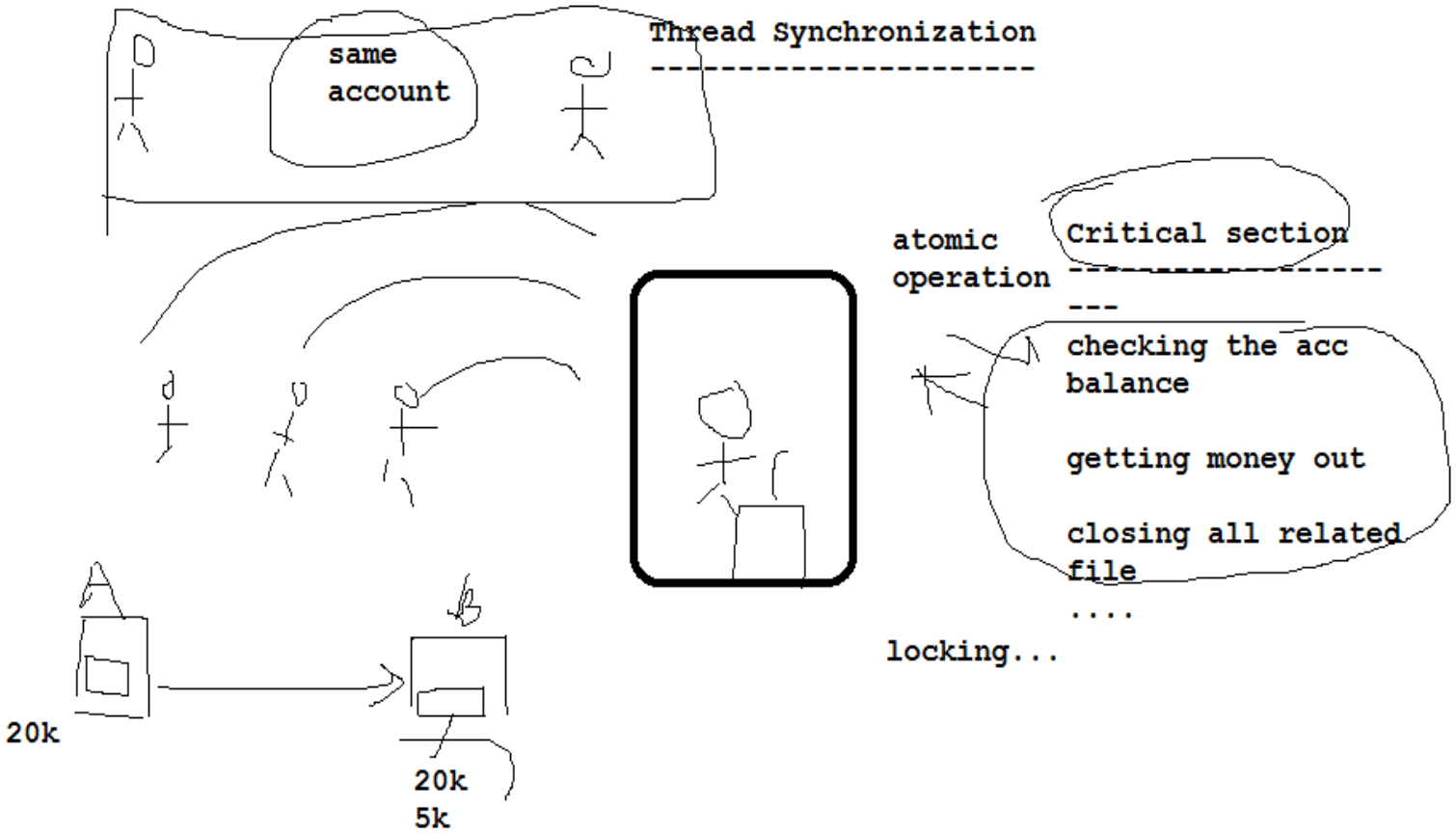
pl wait for me for 5 min else u can go...

Checking thread priorities

```
class Clicker implements Runnable
{
     int click=0;
     Thread t;
     private volatile boolean running=true;
     public  Clicker(int p)
     {
          t=new Thread(this);
          t.setPriority(p);
     }
     public void run()
     {
          while(running)
               click++;
     }
     public void stop()
     {
          running=false;
     }
     public void start()
     {
          t.start();
     }
}
```

```
.....
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
Clicker hi=new Clicker(Thread.NORM_PRIORITY+2);
Clicker lo=new Clicker(Thread.NORM_PRIORITY-2);
lo.start();
hi.start();
try
{
     Thread.sleep(10000);
}
catch(InterruptedException ex){}
lo.stop();
hi.stop();
//wait for child to terminate
try
{
     hi.t.join();
     lo.t.join();
}
catch(InterruptedException ex)
{

}
System.out.println("Low priority thread:"+lo.click);
System.out.println("High priority thread:"+hi.click);
....
....
```

Understanding thread synchronization



Synchronization

- Synchronization
  - Mechanism to controls the order in which threads execute
  - Competition vs. cooperative synchronization

- Mutual exclusion of threads
  - Each synchronized method or statement is guarded by an object.
  - When entering a synchronized method or statement, the object will be locked until the method is finished.
  - When the object is locked by another thread, the current thread must wait.

Using thread synchronization

```
class CallMe
{
     synchronized void call(String msg)
     {
          System.out.print("["+msg);
          try
          {
               Thread.sleep(500);
          }
          catch(InterruptedException ex){}
          System.out.println("]");
     }
}
```

```
class Caller implements Runnable
{
     String msg;
     CallMe target;
     Thread t;
     public Caller(CallMe targ,String s)
     {
          target=targ;
          msg=s;
          t=new Thread(this);
          t.start();
     }
     public void run()
     {
          target.call(msg);
     }
}
```

```
Caller ob1=new Caller(target, "Hello");
Caller ob2=new Caller(target,"Synchronized");
Caller ob3=new Caller(target,"Java");
```

Inter thread communication

- Java have elegant Interprocess communication using wait() notify() and notifyAll() methods
- All these method defined final in the Object class
- Can be only called from a synchronized context

wait() and notify(), notifyAll()

- wait()
  - Tells the calling thread to give up the monitor and go to the sleep until some other thread enter the same monitor and call notify()

- notify()
  - Wakes up the first thread that called wait() on same object

- notifyAll()
  - Wakes up all the thread that called wait() on same object, highest priority thread is going to run first

Incorrect implementation of produce consumer

```
class Q
{
        int n;
        synchronized int get()
        {
            System.out.println("got:"+n);
            return n;
        }
        synchronized void put(int n)
        {
            this.n=n;
            System.out.println("Put:"+n);
        }
}
```

```
class Producer implements Runnable
{
    Q q;
    public Producer(Q q) {
        this.q=q;
        new Thread(this,"Producer").start();
    }
    public void run()
    {
        int i=0;
        while(true)
            q.put(i++);
    }
}
```

```
public class PandC {
public static void main(String[] args) {
    Q q=new Q();
    new Producer(q);
    new Consumer(q);
    System.out.println("ctrol C for exit");
}
}
```

```
class Consumer implements Runnable
{
    Q q;
    Consumer(Q q)
    {
        this.q=q;
        new Thread(this,"consumer").start();
    }
    public void run()
    {
        while(true)
            q.get();
    }
}
```

Correct implementation of produce consumer ...

```
class Q
{   int n;
        boolean valueSet=false;
        synchronized int get()
        {
            if(!valueSet)
                try
                {
                    wait();
                }
            catch(InterruptedException ex){}
            System.out.println("got:"+n);
            valueSet=false;
            notify();
            return n;
        }
        synchronized void put(int n)
        {
            if(valueSet)
                try
                {
                    wait();
                }
            catch(InterruptedException ex){}
            this.n=n;
            valueSet=true;
            System.out.println("Put:"+n);
            notify();
        }
}
```

DAY-5

rgupta.mtech@gmail.com

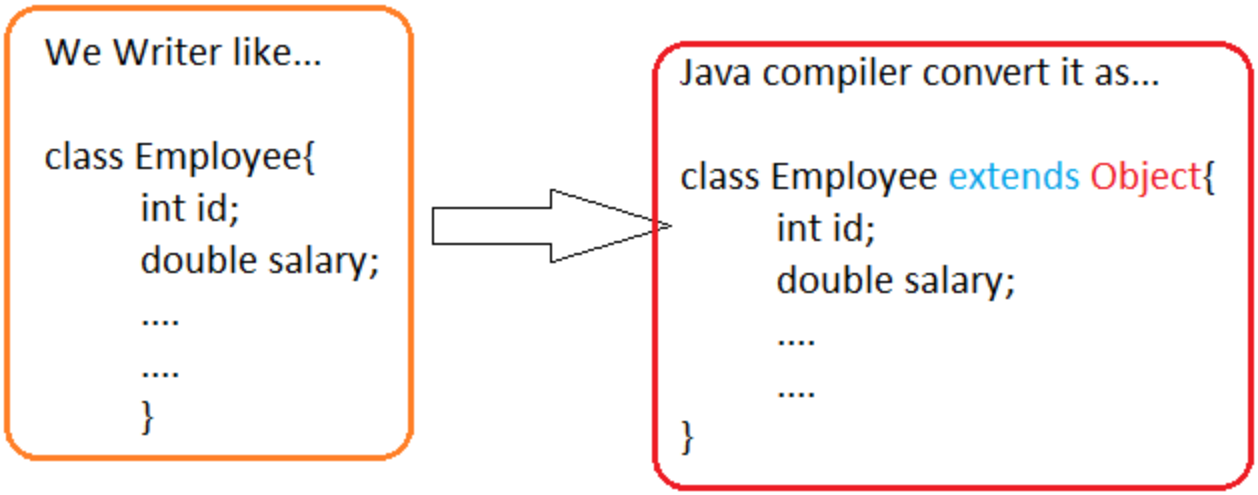
Session-1

- Session-1
  - Object class in Java
  - Collections Framework
  - List, Map, Set usages introduction
  - Usages
    - ArrayList, LinkedList, HashMap, TreeMap, HashSet
  - Comparable, Comparator interface implementation
  - User define key in HashMap

Object

- Object is an special class in java defined in java.lang

this



- Why Java has provided this class?

Method defined in Object class...

- String toString()
- boolean equals()
- int hashCode()
- clone()
- void finalize()
- getClass()
- Method that can’t be overridden
  - final void notify()
  - final void notifyAll()
  - final void wait()

toString( )

- If we do not override toString() method of Object class it print Object Identification number by default
- We can override it to print some useful information….

```
class Employee{
    private int id;
    private double salary;

    public Employee(int id, double salary) {
        this.id = id;
        this.salary = salary;
    }

}

        ......
        ......
        Employee e=new Employee(22, 333333.5);
        System.out.println(e);
        .....
        ....

O/P
-------
com.Employee@addbf1
```

Java simply print object identification number not so useful message for client

toString()

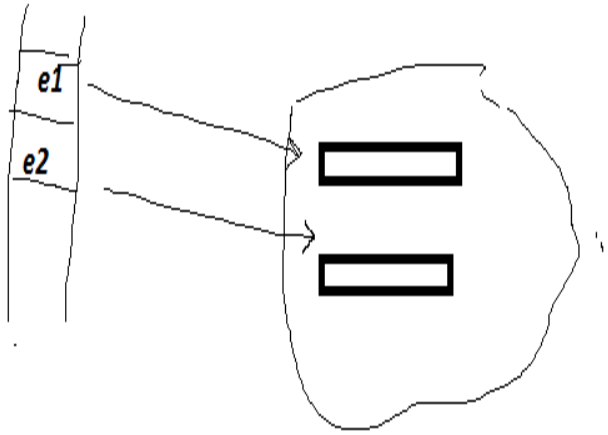
```
class Employee{
     private int id;
     private double salary;

     public Employee(int id, double salary) {
          this.id = id;
          this.salary = salary;
     }

     @Override
     public String toString() {
          return "Employee [id=" + id + ", salary=" + salary + "]";
     }

}
public class DemoToString {
     public static void main(String[] args) {
          Employee e=new Employee(22, 333333.5);
          System.out.println(e);
     }
}
```

equals

- What O/P do you expect

```
Employee e1=new Employee(22, 333333.5);
         Employee e2=new Employee(22, 333333.5);

         if(e1==e2)
             System.out.println("two employees are equals....");
         else
             System.out.println("two employees are not equals....");
```

- O/P would be two employees are not equals.... ???
- Problem is that using == java compare object id of two object and that can never be equals, so we



Overriding equals()

- Don't forget DRY run....

```
@Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Employee other = (Employee) obj;
        if (id != other.id)
            return false;
        if (Double.doubleToLongBits(salary) != Double
                .doubleToLongBits(other.salary))
            return false;
        return true;
    }
```

hashCode()

- Whenever you override equals()for an type don’t forget to override hashCode() method…
- hashCode() make DS efficient
- What hashCode does
  - HashCode divide data into buckets
  - Equals search data from that bucket…

```
@Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + id;
        long temp;
        temp = Double.doubleToLongBits(salary);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        return result;
    }
```

clone()

- Lets consider an object that creation is very complicated, what we can do we can make an clone of that object and use that
- Costly , avoid using cloning if possible, internally depends on serialization
- Must make class supporting cloning by implementing an r

```
class Employee implements Cloneable{
    private int id;
    private double salary;

    public Employee(int id, double salary) {
        this.id = id;
        this.salary = salary;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        // TODO Auto-generated method stub
        return super.clone();
        //can write more code
    }
```

rg

```
}
```

finalize()

- As you are aware ..Java don’t support destructor
- Programmer is free from memory management
- Memory mgt is done by an component of JVM ie called Garbage collector GC
- GC runs as low priority thread..
- We can override finalize() to request java “Please run this code before recycling this object”
- Cleanup code can be written in finalize() method

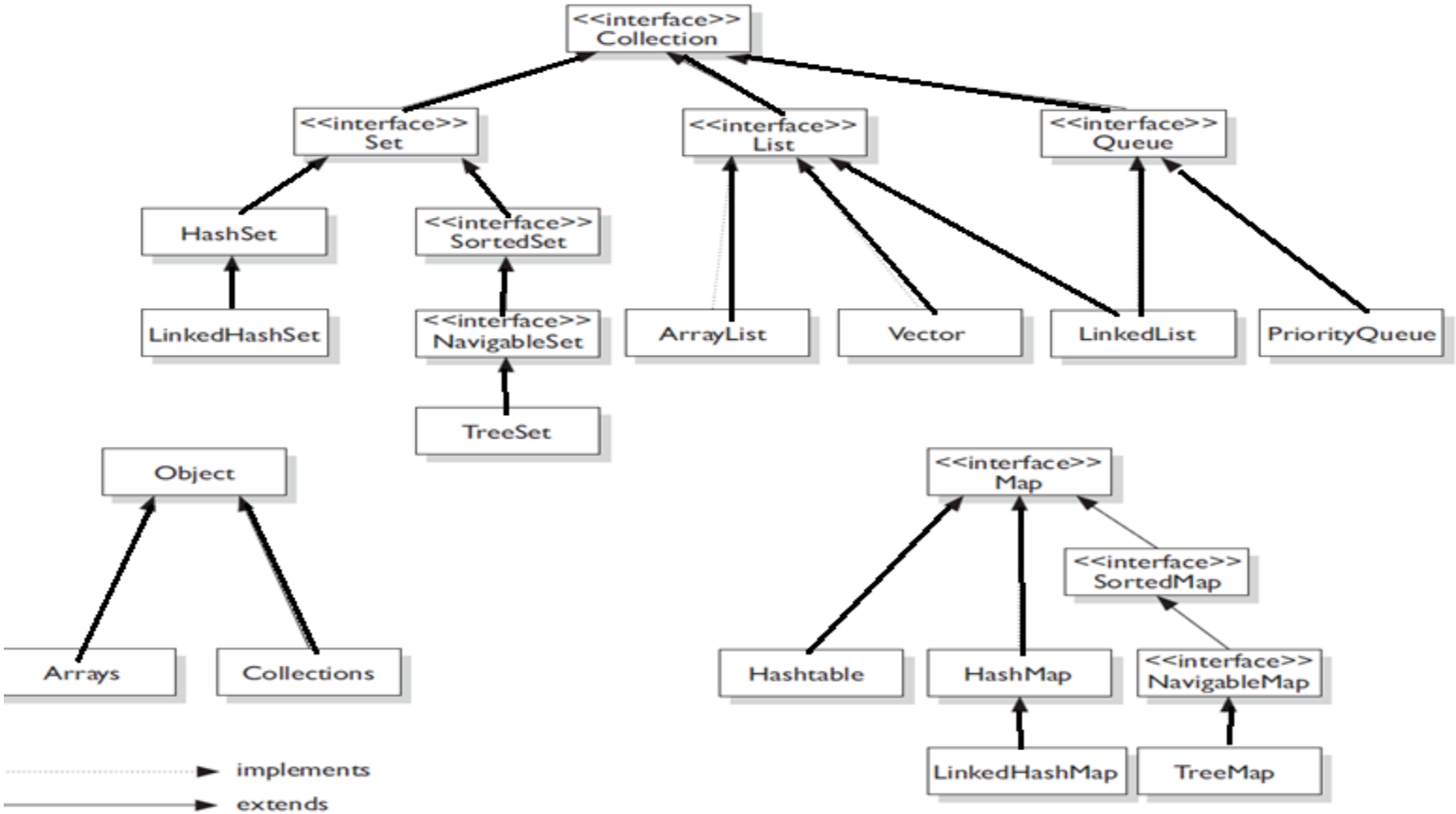
Java collection

- Java collections can be considered a kind of readymade data structure, we should only need to know how to use them and how they work....

  - collection
    - Name of topic
  - Collection
    - Base interface
  - Collections
    - Static utility class provide various useful algorithm

collection

- collection
  - Why it is called an framework?
  - Readymade Data structure in Java...



Four type of collections

Collections come in four basic flavors:

- **Lists** *Lists* of things (classes that implement List).
- **Sets** *Unique* things (classes that implement Set).
- **Maps** Things with a *unique* ID (classes that implement Map).
- **Queues** Things arranged by the order in which they are to be processed.

ArrayList: aka growable

```
List<String>list=new ArrayList<String>();
...
...
list.size();
list.contains("raj");

test.remove("hi");

Collections.sort(list);
```

Note:

```
Collections.sort(list,Collections.reverseOrder());
```

```
Collections.addAll(list2,list1);
```

Add all elements from list1 to end of list2

```
Collections.frequency(list2,"foo");
```

---

print frequency of "foo" in the list2 collection

```
boolean flag=Collections.disjoint(list1,list);
```

---

return "true" if nothing is common in list1 and list2

Sorting with the Arrays Class

```
Arrays.sort(arrayToSort)

Arrays.sort(arrayToSort, Comparator)
```

rgupta.mtech@gmail.com

ArrayList of user defined object

```
class Employee{
        int id;
        float salary;
        //getter setter
        //const
        //toString
        }

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

        list.add(new Employee(121,"rama");
        list.add(new Employee(121,"rama");
        list.add(new Employee(121,"rama");

        System.out.println(list);

        Collections.sort(list);
```

How java can decide how to sort?

rgupta.mtech@gmail.com

Comparable and Comparator interface

- We need to teach Java how to sort user define object
- Comparable and Comparator interface

Comparable	Comparator
java.lang	java.util
Natural sort	seconday sorts
Only one sort sequence is possible	as many as you want
need to change the design of the class	Dont need to change desing of the class
need to override	need to override
public int compareTo(Employee o)	public int compare(Employee o1, Employee o2)

Implementing Comparable

```
class Employee implements Comparable<Employee>{
    private int id;
    private double salary;
    .......
    .......
    .......

        @Override
    public int compareTo(Employee o) {
        // TODO Auto-generated method stub
        Integer id1=this.getId();
        Integer id2=o.getId();
        return id1.compareTo(id2);
    }
```

Comparator

- Don't need to change Employee class

```
class SalarySorter implements Comparator<Employee>{

    @Override
    public int compare(Employee o1, Employee o2) {
        // TODO Auto-generated method stub
        Double sal1=o1.getSalary();
        Double sal2=o2.getSalary();

        return sal1.compareTo(sal2);
    }

}
```

Useful stuff

Converting Arrays to Lists

```
String[] sa = {"one", "two", "three", "four"};
List sList = Arrays.asList(sa);
```

Converting Lists to Arrays

```
List<Integer> iL = new ArrayList<Integer>();

for(int x=0; x<3; x++)
iL.add(x);

Object[] oa = iL.toArray(); // create an Object array

Integer[] ia2 = new Integer[3];

ia2 = iL.toArray(ia2); // create an Integer array
```

Arrays.binarySearch(arrayFromWhichToSearch,"to search"))

---

return -ve no if no found

array must be sorted before hand otherwise o/p is not predictiale

rgupta.mtech@gmail.com

Useful examples...

user define funtion to print the arraylist/linkedlist

```
----------------------------------------------------------

printMe(list1);
....
.....


public void printMe(list<String>list){

        for(String s:list)
        {
                Sysout(s);
        }
}
```

user define funtion to remove stuff from a arraylist /linkedlist

```
-----------------------------------------------------------------
removeStuff(list,2,5);


...
...

public void removeStuff(List<String>l, int from, int to)
{
        l.subList(from,to).clear();
}
```

rgupta.mtech@gmail.com

Useful examples...

Merging two link lists

```
ListIterator ita=a.listItrator();
Iteratory itb=b.iterator();

while(itb.hasNext())
{
        if(ita.hasNext())
                ita.next();

        ita.add(itb.next());
}
```

Removing every second element from an linkedList

```
itb=b.iterator();

while(itb.hasNext())
{
        itb.next();

        if(itb.hasNext())
        {
                itb.next();

                itb.remove();
        }
}
```

LinkedList : AKA Doubly Link list.....
can move back and forth.....

Imp methods
---

boolean hasNext()
Object next()
boolean hasPrevious()
Object previous()

More methods
---

void addFirst(Object o);

void addLast(Object o);

object getFirst();

object getLast();

add(int pos,Object o);

rgupta.mtech@gmail.com

Useful examples...

```
user define funtion to print linkedlist in reverse order
-----------------------------------------------------------

reversePrint(list);
...
....

public void reversePrint(list<String>l ){

        ListIterator<String>it=l.iterator(l.size());

        while(it.hasPrevious())
                Sysout(it.previous());
}
```

fundamental diff bw ArrayList and LinkedList

- ArrayLists manage arrays internally.
  [0][1][2][3][4][5] ....
- List<Integer> arrayList = new ArrayList<Integer>();

- LinkedLists consists of elements where each element has a reference to the previous and next element
  [0]->[1]->[2] ....
  <- <-

ArrayList vs LinkedList

- Java implements ArrayList as array internally
  - Hence good to provide starting size
    - i.e. List<String> s=new ArrayList<String>(20); is better then List<String> s=new ArrayList<String>();
- Removing element from starting of arraylist is very slow?
  - list.remove(0);
  - if u remove first element, java internally copy all the element (shift by one)

rgupta.mtech@gmail.com

- Adding element at middle in ArrayList is very

Performance ArrayList vs LinkedList !!!

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class App
{
 public static void main(String[] args)
        {
        List<Integer> arrayList = new ArrayList<Integer>();
        List<Integer> linkedList = new LinkedList<Integer>();

         doTimings("ArrayList", arrayList);
         doTimings("LinkedList" , linkedList);
   }
```

```
private static void doTimings(String type, List<Integer> list)
  {
    for(int i=0; i<1E5; i++)
          list.add(i);
    long start = System.currentTimeMillis();

    /*
    // Add items at end of list
    for(int i=0; i<1E5; i++)
        {
      list.add(i);
    }
    */

    // Add items elsewhere in list
    for(int i=0; i<1E5; i++)
        {
      list.add(0, i);
    }
    long end = System.currentTimeMillis();

    System.out.println("Time taken: " + (end - start) + " ms for " + type);
  }
}
```

```
Time taken: 7546 ms for ArrayList
Time taken: 76 ms for LinkedList
```

rguptamtech@gmail.com

HashMap

- Key ---->Value
- declaring an hashmap
  - HashMap<Integer, String> map = new HashMap<Integer, String>();
- Populating values

```
map.put(5, "Five");
 map.put(8, "Eight");
map.put(6, "Six");
map.put(4, "Four");
map.put(2, "Two");
```

- Getting value
  - String text = map.get(6);
  - System.out.println(text);

Looping through HashMap

```
for(Integer key: map.keySet())
      {
         String value = map.get(key);

         System.out.println(key + ": " + value);
       }
```

most imp thing to remember

---

order of getting key value is not maintained

ie hashMap dont keep key and value in any particular order

Other map varients

- LinkedHashMap
  - Aka. Doubly link list
  - key and value are in same order in which you have inserted.....

- TreeMap
  - sort keys in natural order(what is natural order?)
  - for int
    - 1,2,3.....
  - for string
    - "a","b".....
  - For user define key

set

- Don’t allow duplicate element

three types:

---

hashset
linkedhashset
treeset

HashSet does not retain order.

```
Set<String> set1 = new HashSet<String>();
```

LinkedHashSet remembers the order you added items in

```
Set<String> set1 = new LinkedHashSet<String>();
```

TreeSet sorts in natural order

```
Set<String> set1 = new TreeSet<String>();
```

Printing freq of unique words
from a file in increseing order of freq

---

words	freq
Apple	7
Ball	5
...	

rgupta.mtech@gmail.com

User define key in HashMap

- If you are using user define key in HashMap do not forget to override hashcode for that class
- Why?
  - We may not find that content again !

HashMap vs Hashtable

- Hashtable is threadsafe, slow as compared to HashMap
- Better to use HashMap

- Some more intresting difference
  - Hashtable give runtime exception if key is “null” while HashMap don’t

Session-2

- Java 5 Language Features II
  - Generics
  - Annotations

rgupta.mtech@gmail.com

Generics

- Before Java 1.5
  - List list=new ArrayList();
    - Can add anything in that list
    - Problem while retrieving

- Now Java 1.5 onward
  - List<String> list=new ArrayList<String>();
    list.add("foo");//ok
    list.add(22);// compile time error

  - Generics provide type safety
  - Generics is compile time phenomena…

Issues with Generics

- Try not to mix non Generics code and Generics code…we can have strange behaviour.

```
package com;
import java.util.*;

public class DemoGen1 {
    public static void main(String[] args) {

        List<String> list=new ArrayList<String>();
        list.add("foo");
        list.add("bar");

        strangMethod(list);

        for(String temp:list)
            System.out.println(temp);
    }

    private static void strangMethod(List list) {
        list.add(new Integer(22));// OMG.....
    }

}
```

Polymorphic behaviour

```
class Animal {
}

class Cat extends Animal{
}

class Dog extends Animal{
     }
```

```
Animal []aa=new Cat[4];// allowed

List<Animal>list=new ArrayList<Cat>()
```

rgupta.mtech@gmail.com

<? extends XXXXXX>

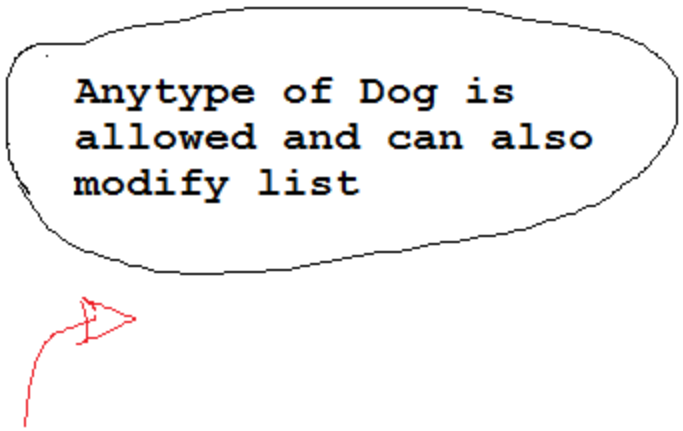
```
package com;
import java.util.*;

public class DemoGen1 {
    public static void main(String[] args) {

        List<Integer> list=new ArrayList<Integer>();
        list.add(22);
        list.add(33);

        strangMethod(list);

        for(Integer temp:list)
            System.out.println(temp);
    }

    private static void strangMethod(List<? extends Number> list) {
        list.add(new Integer(22));//Compile time error..... Good
    }

}
```

in strangMethod() we can pass any derivative of Number class but we are not allowed to modify the list

rgupta.mtech@gmail.com

<? Super XXXXX>

```
class Animal {
}

class Cat extends Animal{
}

class Dog extends Animal{
     }

class CostlyDog extends Dog{
}
     ........
     .........
          List<Dog>list=new ArrayList<Dog>();
          list.add(new Dog("white"));
          list.add(new Dog("red"));
          list.add(new Dog("black"));
          strangMethod(list);

     private static void strangMethod(List<? super Dog> list) {
          list.add(new CostlyDog());
     }

     .........
     ......
```



Generic class

```
class MyObject<T>{
     T myObject;

     public T getMyObject() {
          return myObject;
     }

     public void setMyObject(T myObject) {
          this.myObject = myObject;
     }

}
public class GenClass {

          public static void main(String[] args) {
               MyObject<String> o=new MyObject<String>();
               o.setMyObject(new Integer(22));
               //System.out.println(it.intValue());
          }
}
```

will not compile !!!

rgupta.mtech@gmail.com

Generic method

```
class MaxOfThree{

     public static <T extends Comparable<T>> T maxi(T a,T b, T c){
          T max=a;
          if(b.compareTo(a)>0)
               max=b;
          if(c.compareTo(max)>0)
               max=c;
          return max;
     }

}
```

Annotation

- @ magic !!!
- Annotations in Java is all about adding meta-data facility to the Java Elements.

- Like Classes, Interfaces or Enums, Annotations define a type in Java and they can be applied to several Java Elements.

- Tools which will read and interpret the Annotations will implement a lot of functionalities from the meta-information obtained.

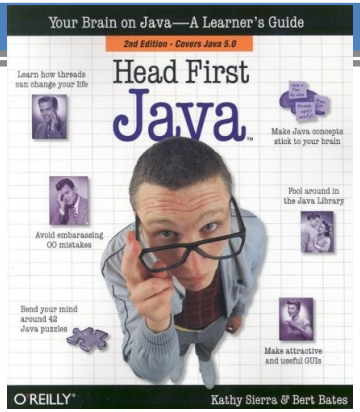
- For example, they can ensure the consistency between classes, can check the validity of the paramters passed by the clients at run-time and can generate lot of base code for a project.

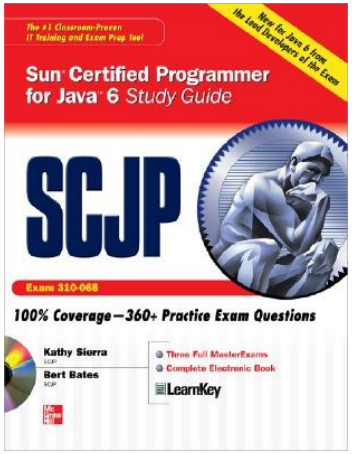
Built-in Annotations in Java

- There are some pre-defined annotations available in the Java Programming language. They are,
  - Override
  - Deprecated
  - SuppressWarnings

- Demo user define annotation and its need...

References:-





rgupta.mtech@gmail.com