Object Oriented Analysis and Design with Java

UE20CS352

Prof . Priya Badarinath
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

UE20CS352: Object Oriented Analysis and Design with Java

Unit 2 : Introduction to Object Oriented Programming

Prof . Priya Badarinath

Department of Computer Science and Engineering

Agenda-

- Introduction to Java
- Abstraction
- Encapsulation
- Composition

Introduction to Java

- Java programming language was originally developed by Sun Microsystems, by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0)

- The new J2 versions were renamed as Java SE, Java EE and Java ME respectively.

1. Java Platform, Standard Edition (Java SE)
2. Java Platform, Enterprise Edition (Java EE)
3. Java Platform, Micro Edition (Java ME)

- Java is guaranteed to be Write Once, Run Anywhere.

- Java was mainly developed to create software for consumer electronic devices that could be controlled by a remote.

Introduction to Java: Features

1. Simple

There is no need for header files, pointer arithmetic, structures, unions, operator overloading, virtual base classes

2. Object-Oriented

Object-oriented design is a programming technique that focuses on the data (= objects) and on the interfaces to that object.

3. Distributed

Java has an extensive library of routines for coping with TCP/IP protocols like HTTP and FTP. Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.

4. Robust

Inbuilt exception handling features and memory management features.

5. Secure

Java is intended to be used in networked/distributed environments. Toward that end, a lot of emphasis has been placed on security. Java enables the construction of virus-free, tamper-free systems.

Introduction to Java: Features

6.Portable

Unlike C and C++, there are no “implementation-dependent” aspects of the specification. The sizes of the primitive data types are specified,nas is the behavior of arithmetic on them.

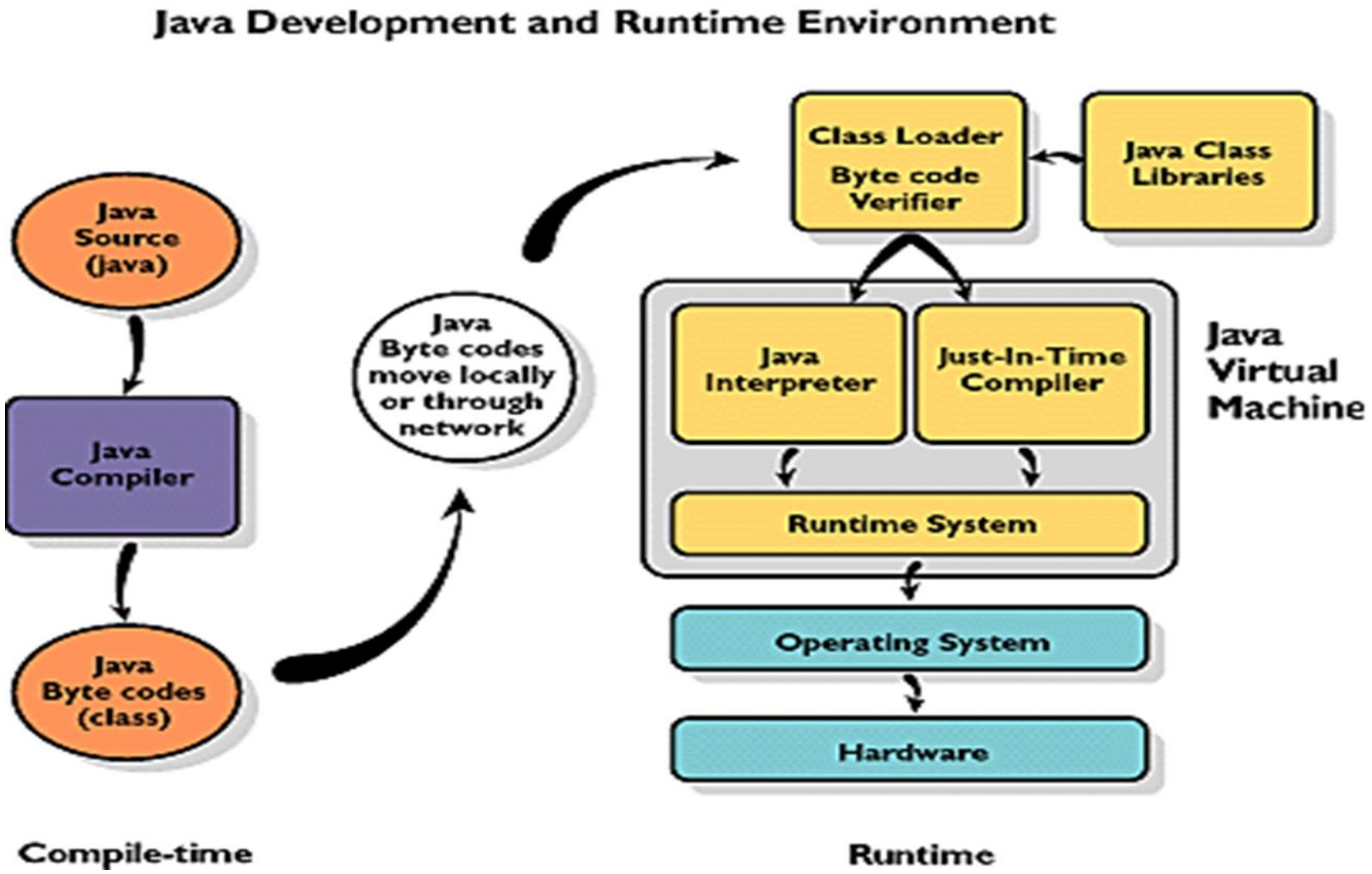
7. Interpreted

Programmer writes code that will be executed by an interpreter, rather than compiled into object code loaded by the OS and executed by CPU directly. An interpreter executes the code line by line.

8. High-Performance

The bytecodes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on. The just-in-time compiler knows which classes have been loaded. It can use inlining when, based upon the currently loaded collection of classes, a particular function is never overridden, and it can undo that optimization later if necessary.

Java Runtime Environment(JRE)

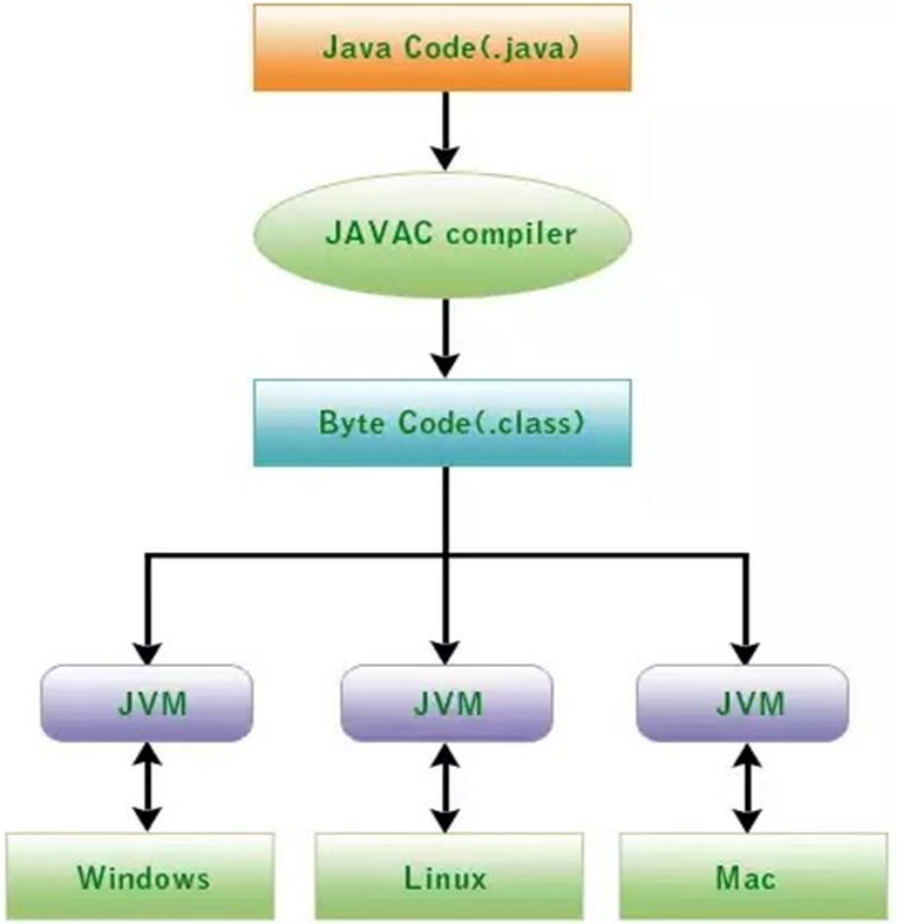


JAVA Virtual Machine

JVM is a platform-independent execution environment that converts Java bytecode(.class file) into machine language and executes it.

It is:

- **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
- **An implementation** Its implementation is known as JRE (Java Runtime Environment).
- **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.



Concept: Data Encapsulation

- The wrapping up of data and functions into a single unit is known as encapsulation.
- The data is not accessible to the outside world, only those function which are wrapped in can access it.
- These functions provide the interface between the object's data and the program.
- This insulation of the data from direct access by the program is called data hiding or information hiding.

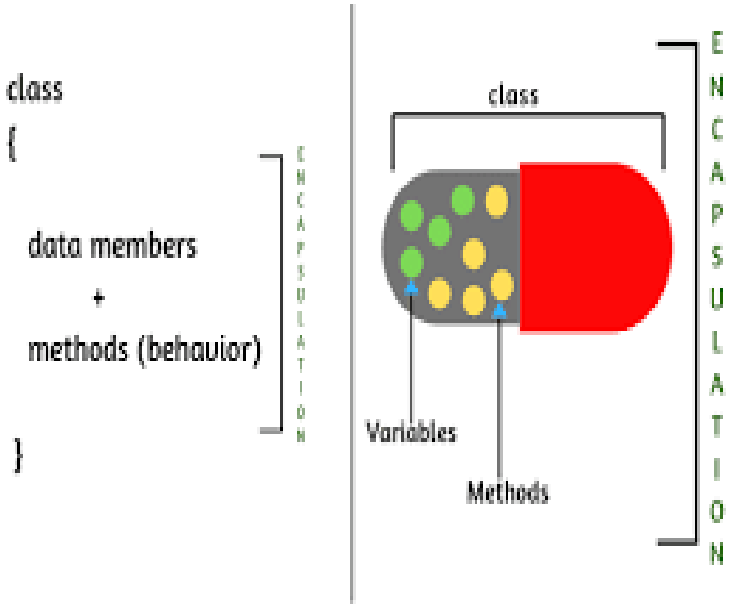
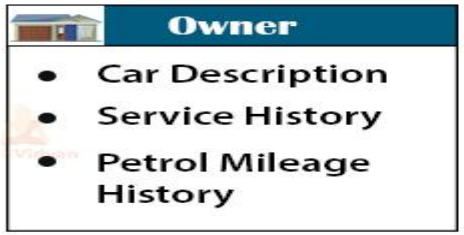
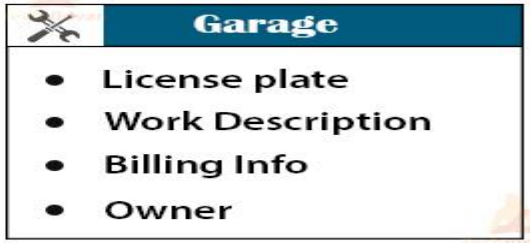


Fig: Encapsulation

Concept: Data Abstraction

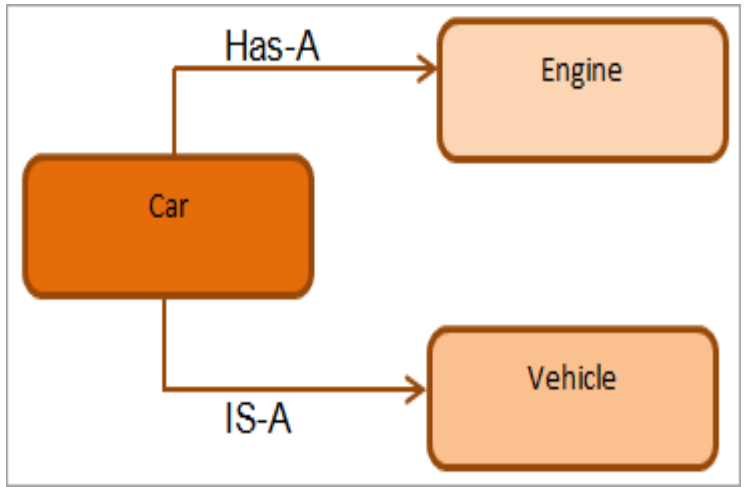
- Abstraction refers to the act of representing essential features without including the background details or explanations.
- Since classes use the concept of data abstraction, they are known as **Abstract Data Types (ADT)**





Concept: Composition

- The Composition is a way to design or implement the "has-a" relationship.
- Composition and Inheritance both are design techniques.
- The Inheritance is used to implement the "is-a" relationship. The "has-a" relationship is used to ensure the code reusability in our program.
- In Composition, we use an instance variable that refers to another object.



THANK YOU

Priya Badarinath
Department of Computer Science and Engineering

priyab@pes.edu

Object Oriented Analysis and Design using Java (UE20CS352)

Prof. Mahitha G
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

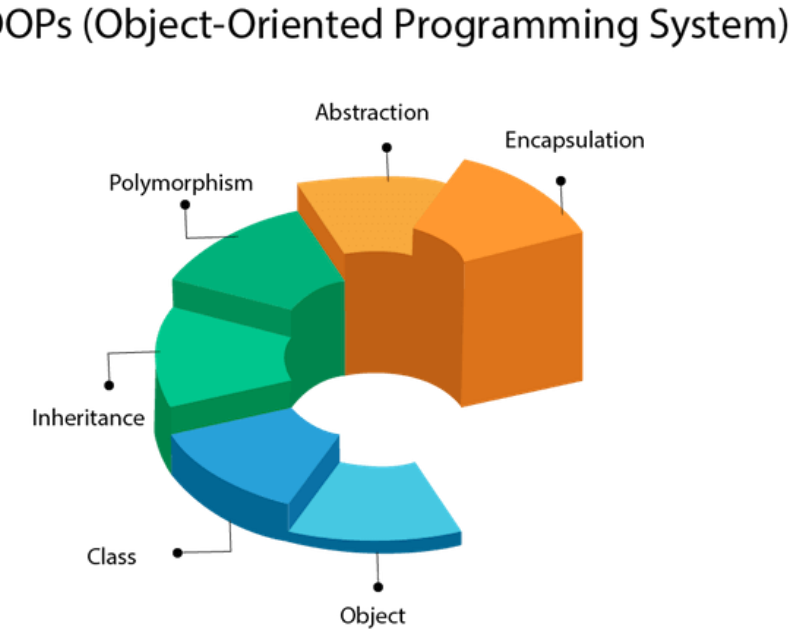
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Introduction to OO Programming

Prof. Mahitha G
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

Department of Computer Science and Engineering



PES UNIVERSITY

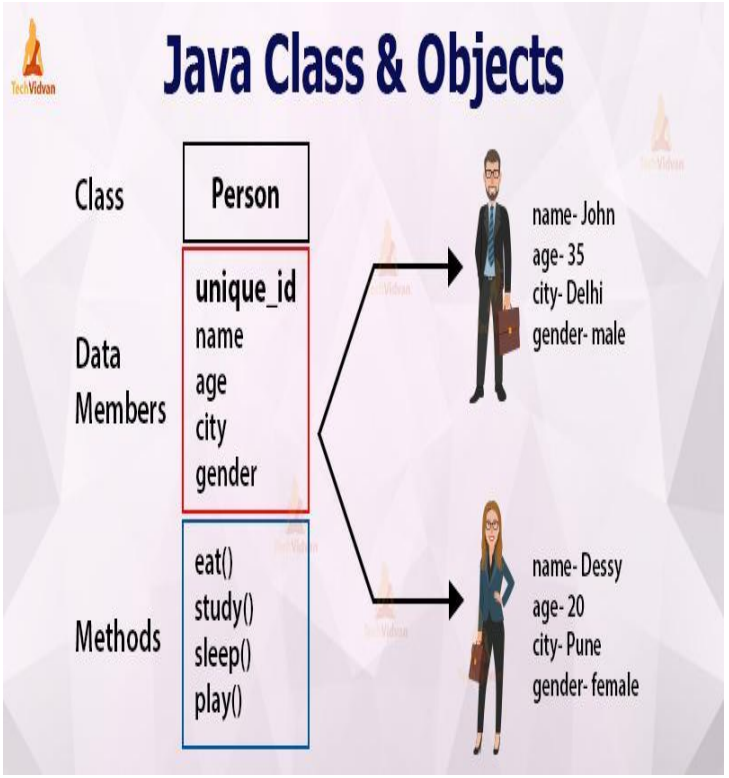
T1 : Chapter 6: Introducing Classes

1: “Java the Complete Reference”, Herbert Schildt ,McGraw-Hill ,11th Edition, 2018.

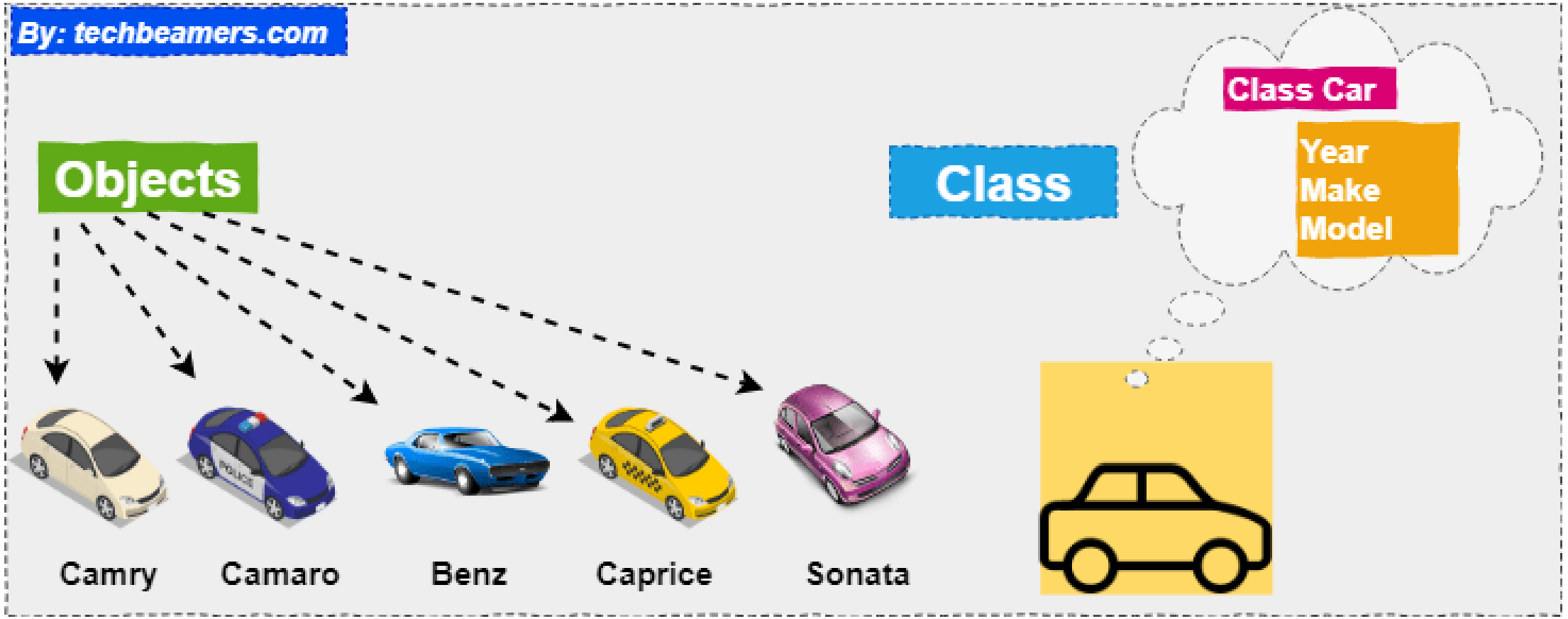
Object Oriented Programming: Classes and Objects

CLASSES and OBJECTS

- Defines a new data type.
- **Class is a template** for an object and an **object is an instance of a class.**
- A class may contain only data or only code or both.
- A class is the template or blueprint from which objects are made
- When you construct an object from a class, you are said to have created an instance of the class.
- Any concept to be implemented in java must be encapsulated within a class.
- **Think about classes as cookie cutters. Objects are the cookies themselves.**



Object Oriented Programming: Class and Objects



Object Oriented Analysis and Design using Java

Object Oriented Programming: Class and Objects

General definition of a class in java:

class class_name {

data_type instance_variable1;

data_type instance_variable2;……..

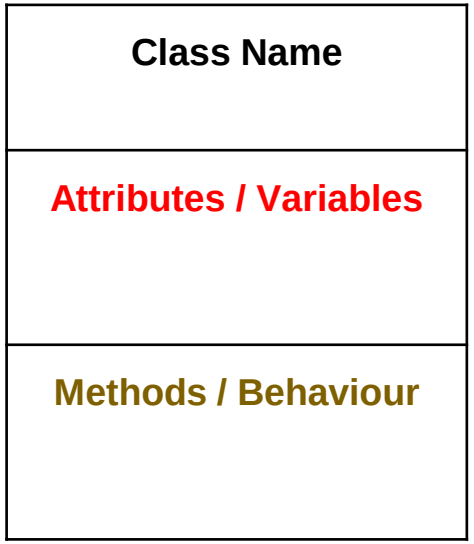
data_type method1() {…//body of the method}

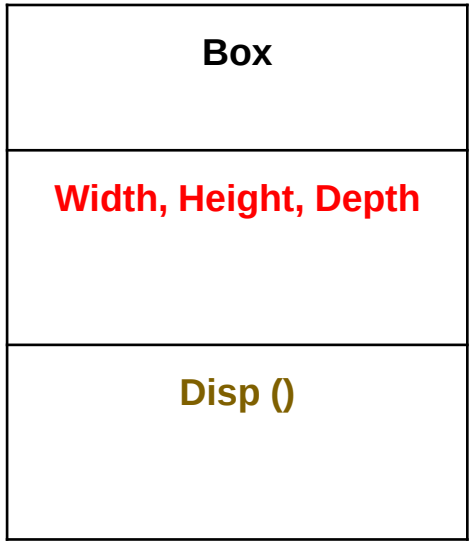
data_type method1() {…//body of the method}

…….

}

Classes contain instance variables and methods





Object Oriented Programming: Class and Objects

To work with OOP, you should be able to identify three key characteristics of objects

1. **The object's behavior-** What can you do with this object, or what methods can you apply to it?
2. **The object's state**—how does the object react when you invoke those methods?
3. **The object's identity**—how is the object distinguished from others that may have the same behavior and state?

- All objects that are instances of the same class share a family resemblance by supporting the same behavior.

- The behavior of an object is defined by **the methods that you can call.**

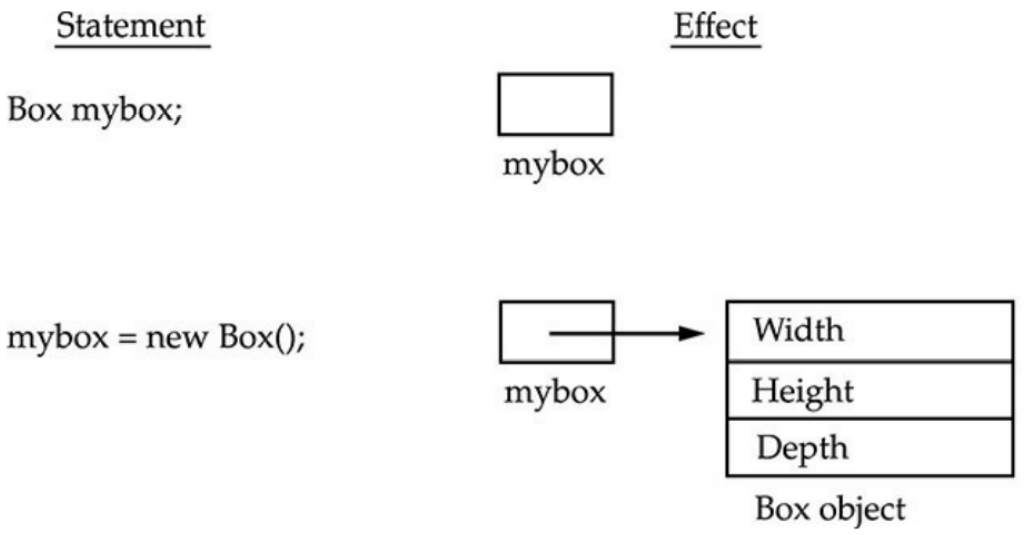
Object Oriented Programming: Class and Objects - Example

Creation of a Class:

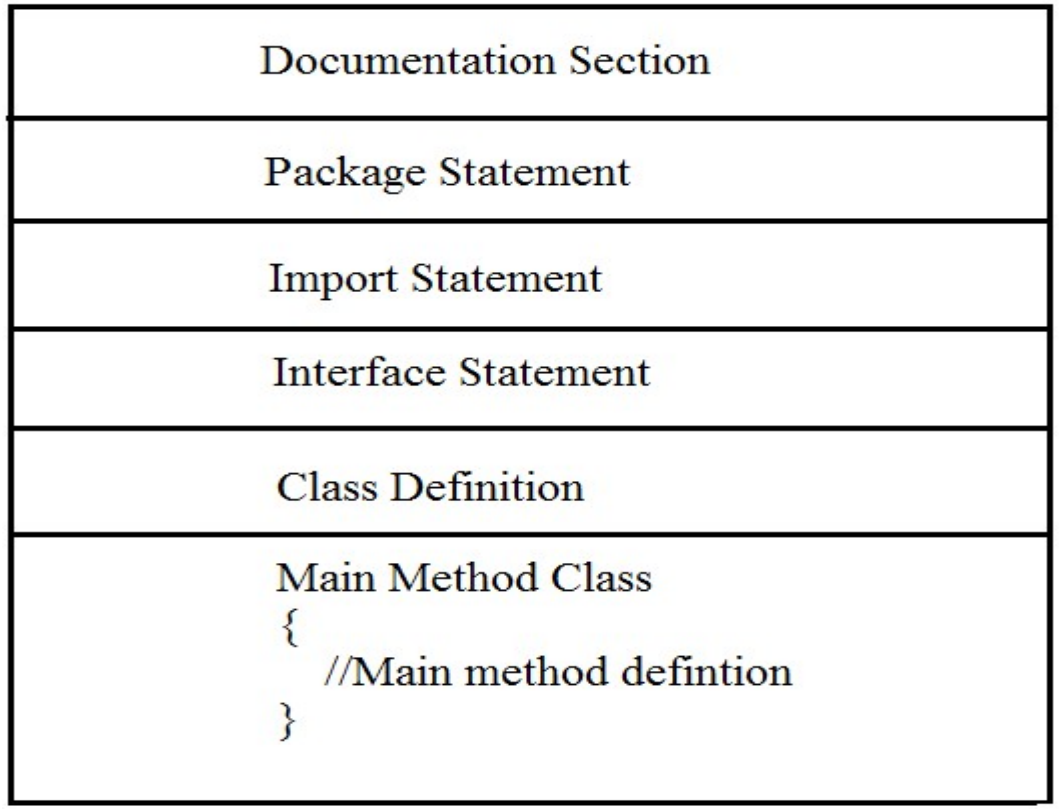
```
class Box
{
        double width;
        double height;
        double depth;
        void disp()
        {
                System.out.println("width: "+width);
                System.out.println("height: "+height);
                System.out.println("depth: "+depth);
        }
}
```

Instantiation of an object:

Box mybox = new Box()



Object Oriented Programming: General Java Program Structure



Object Oriented Programming: Simple Java Program

```
public class FirstProgram
{
   public static void main(String[ ] args)
   {
      System.out.println("Hello Welcome to OOADJ class");
   }
}

//save this as FirstProgram.java
```

Object Oriented Programming: Java Program Execution

Compiling and launching a Java program from the **command line** once JDK is installed.

1: Open command prompt window and navigate to the location of the java program (example: FirstProgram.java).

2: Type 'javac FirstProgram.java' -- to compile your code.

If no errors, the command prompt will take you to the next line

3: Type ' java FirstProgram ' to run your program.

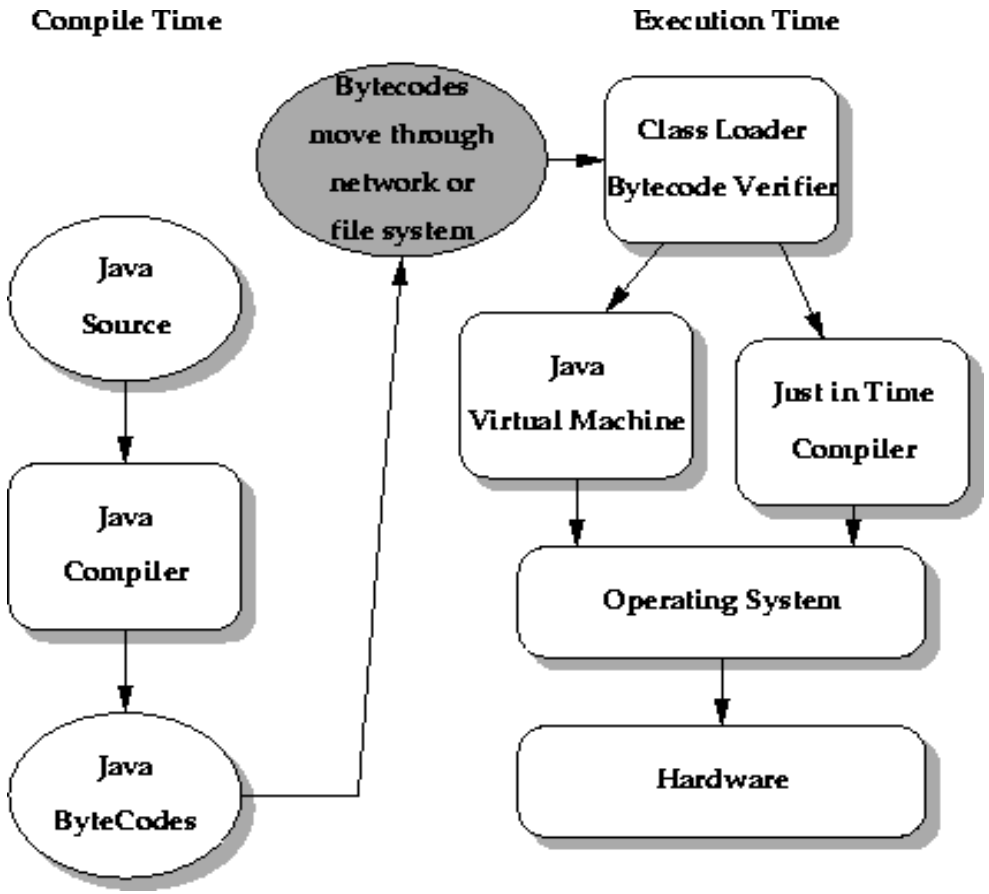
Output is displayed

(Or)

Use IDE – Eclipse, NetBeans, IntelliJ, BlueJ…..

Object Oriented Analysis and Design using Java

Object Oriented Programming: Translation Process



Object Oriented Analysis and Design using Java

Object Oriented Programming: Access Modifiers

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Note: Examples for Default and Protected will be discussed later

Object Oriented Programming: Access Modifiers Private- Example

```
class A{
private int data;
private void msg()
{
data++; //no error
System.out.println("Welcome to OOAD with Java class");
}
  }
public class Sample
{
 public static void main(String args[])
{   A obj=new A();
   System.out.println(obj.data);                //Compile Time Error
   obj.msg();                                   //Compile Time Error
   }
}
```

//accessing the private members, outside the class, so -- a compile-time error.

Object Oriented Analysis and Design using Java

Object Oriented Programming: Access Modifiers Public - Example

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
class A{
public  int data;
public void msg()
{
data++; //no error
System.out.println("Welcome to OOAD with Java class");
}
  }
public class Sample
{
 public static void main(String args[])
{   A obj=new A();
   System.out.println(obj.data);            //no error
   obj.msg();                                //no error
   }
}
```

object
minicooper

Car
class

Color	startCar()
Weight	changeGear()
Speed	slowDown()
Model	brake()

Data members (Attributes)
Common features

method
Common functionalities

>>Swipe

Abstraction

ATM machine

Check balance

Withdraw cash

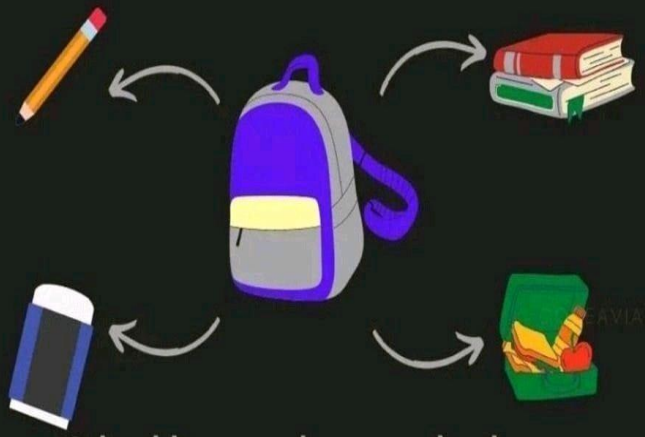
Deposit cash

Print bill

Even though it performs a lot of actions
it doesn't show us the process
It has hidden its process by showing only the main things
like getting inputs and giving the output.

>>Swipe

Encapsulation



School bag can keep our book,
pen, erasers, sharpner so on..

>>Swipe

Inheritance

Mom and daughter

Some properties of mom inherits by her daughter

>>Swipe

Polymorphism

In school
behave like a student

In home
behave like a son

In bus
behave like a passenger

In shopping mall
behave like a customer

>>Swipe

PES UNIVERSITY

THANK YOU

Prof. Mahitha G (mahithag@pes.edu)
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java(UE20CS352)

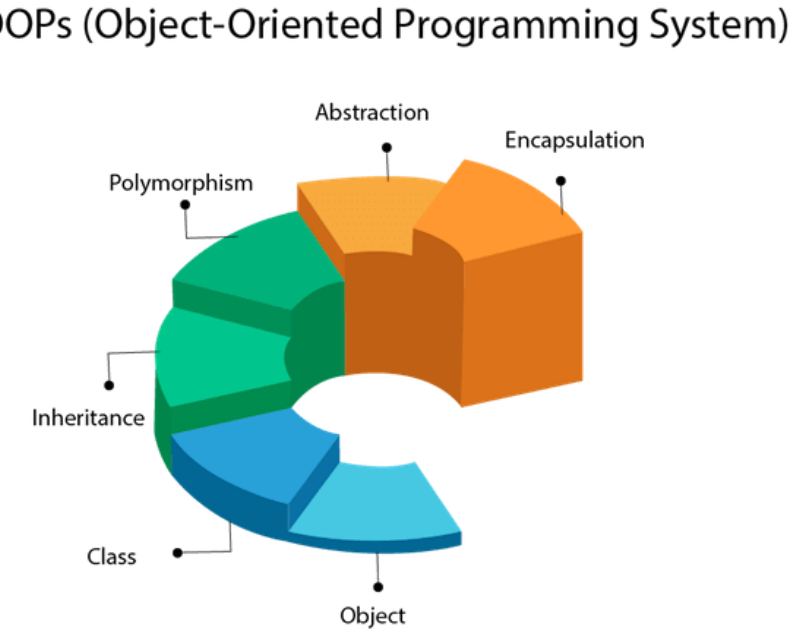
Prof. Mahitha G
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Introduction to OO Programming

Prof. Mahitha G
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S
Department of Computer Science and Engineering



T1 : Chapter 6: Introducing Constructors

1: “Java the Complete Reference”, Herbert Schildt ,McGraw-Hill ,11th Edition, 2018.

Object Oriented Programming: Constructor

Constructor

- A constructor **initializes an object** when it is created.
- It has the **same name as its class** and is syntactically similar to a method.
- Constructors have **no explicit return type**.
- Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other **start-up procedures** required to create a fully formed object.
- All classes have constructors, whether you define one or not, because Java automatically provides a **default constructor** that initializes all member variables to zero or corresponding default value. However, once you define your own constructor, the default constructor is no longer added.
- **Each time a object is created using new operator, constructor is invoked to assign initial values to the data members of the class.**

Object Oriented Programming: Syntax of a default Constructor

```
Class Student
{
Student( )
{
// initialization
  }
}
```

Next we create an object of the above class.

```
Student obj = new Student( );
```

Object Oriented Analysis and Design using Java

Object Oriented Programming: Types of Constructors

Default constructor :

- A constructor that has no parameters. If we don't define a constructor for a class, then compiler creates a default constructor.

- Default constructor provides default values to the objects like 0, false, null etc depending on the data type of the instance variables.

Parameterized constructor:

- A constructor with parameters.
- To initialize the fields of a object with given values
- There are no return value statements in a constructor but constructors return the current class instance.

Copy Constructor: Use to create an exact copy of the existing object

Example programs: CttDemo.java and CttDemo1.java

Object Oriented Programming: Access Modifiers - Example

```
class rect
{
 int l;   int b;
 rect ()
{
System.out.println("ctt");
}
void disp()
{
System.out.println("disp");
}}
public class demo
{
public static void main(String args[])
{
rect r=new rect();
//r.rect();
r.disp();
System.out.println(r.l);
System.out.println(r.b);
}}
```

Object Oriented Programming: Access Modifiers - Example

If the constructor is made private, you cannot create the instance of that class from outside the class.

By default the access modifier is “default”

```
class A{
private A()  { }                          //private constructor
void msg(){System.out.println(“Welcome to OOAD with java class");}
}
public class Sample
{
 public static void main(String args[]){
   A obj=new A();                                  //Compile Time Error
 }
}
```

Object Oriented Programming: Garbage Collector

- Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space.
- One of the best feature of java programming language is the **automatic garbage collection**, unlike other programming languages such as C where memory allocation and de-allocation is a manual process.
- **Garbage Collector** is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program.
- All these unreferenced objects are deleted and space is reclaimed for allocation to other objects.

Object Oriented Programming: finalization

There are certain actions to be performed before an object is destroyed like:

- Closing all the database connections or files
- Releasing all the network resources
- Other Housekeeping tasks
- Recovering the heap space allocated during the lifetime of an object
- Release of release locks

Java provides a mechanism called finalization to do this through finalize( ) method.

Object Oriented Programming: finalize ( )

General form of finalize ( ) method

```
protected void finalize( )
{
//finalization code here
//specify those actions that must be performed before an object is destroyed.
}
```

- Java run time calls this method whenever it is about to recycle an object of the class.
- Keyword protected is used to prevent access to finalize ( ) by the code defined outside its class.
- Called just prior to garbage collection and not called when an object goes out of scope

THANK YOU

Prof. Mahitha G (mahithag@pes.edu)
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java - UE20CS352

Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java
UE20CS352

Parameter Passing –
Value Types and Reference Types

Prof. Priya Badarinath
Department of Computer Science and Engineering

Agenda

1. Introduction
2. Value types with Examples
3. Reference types with Examples

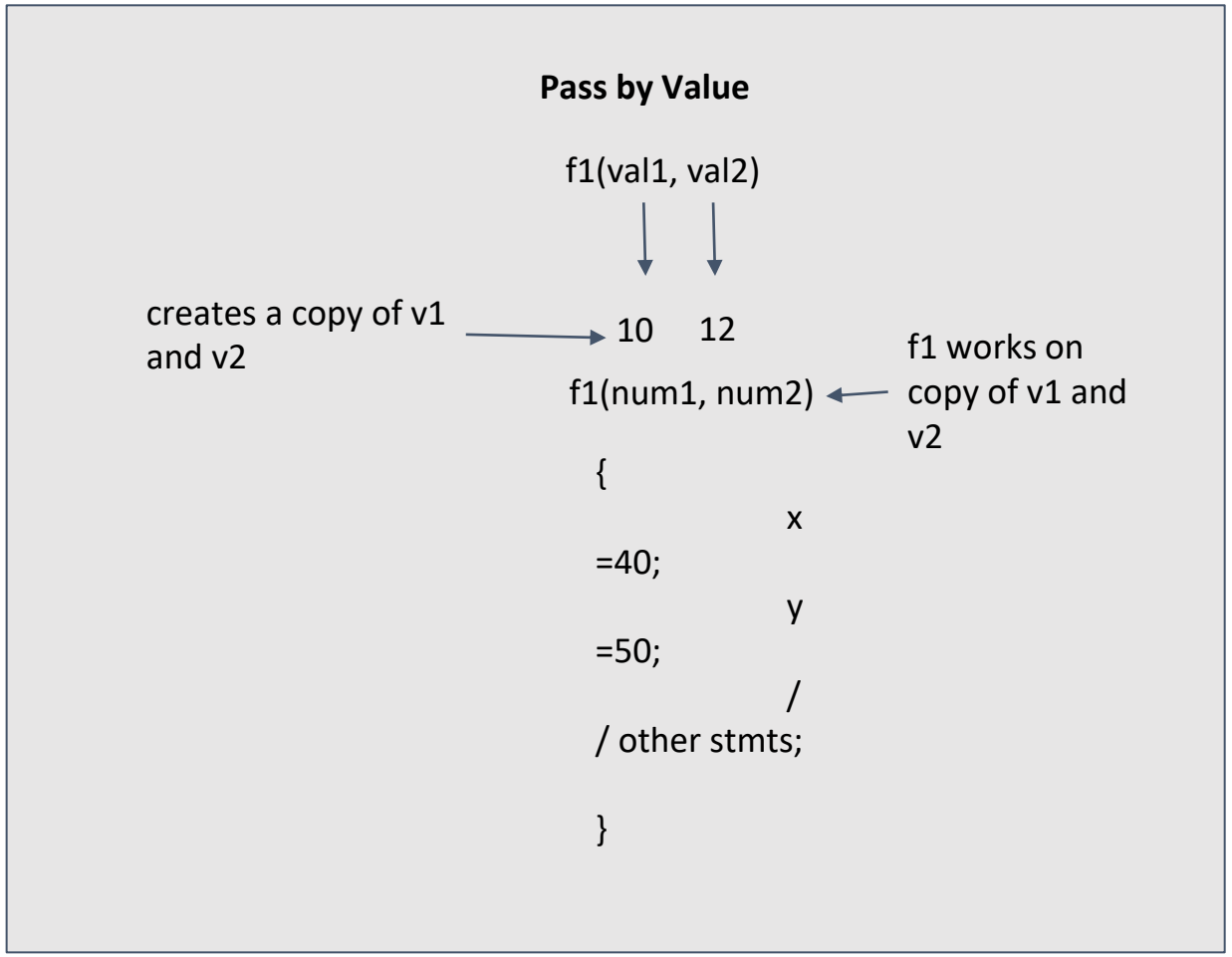
Parameter Passing

Introduction

- Argument is copied to the parameter when some data has to be passed between methods / functions.
- 2 Types of Parameters
  - **Formal Parameter**
  - **Actual Parameter**
- Parameter passing techniques
  - **Pass by Value**
  - **Pass by Reference**

Value types

- Changes made to formal parameter do not get transmitted back to the caller.
- Any modifications to the formal parameter variable inside the called function or method affect only the separate storage location and will not be reflected in the actual parameter in the calling environment.

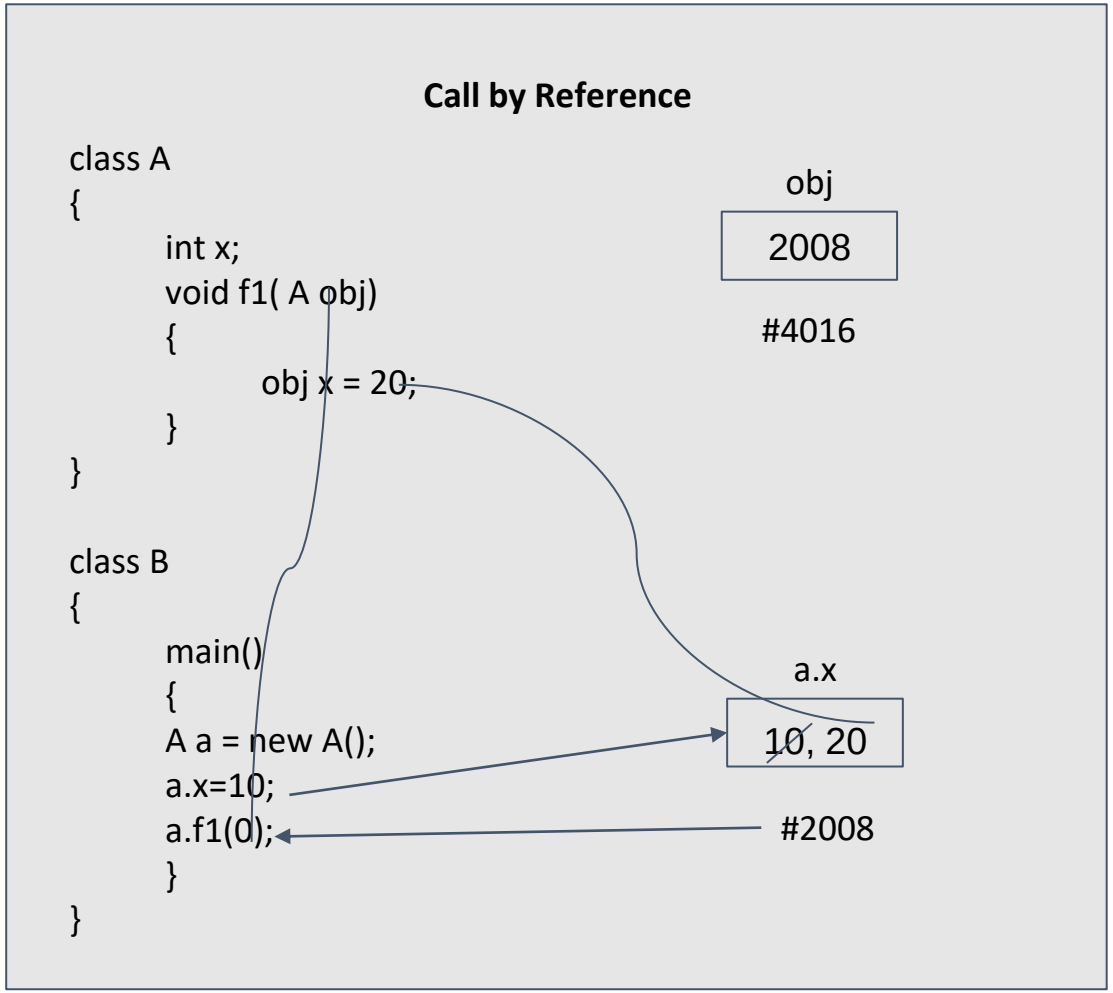


Object Oriented Analysis and Design with Java

Parameter Passing

References- alias

- **Non-Primitive types are references.**
- Changes made to formal parameter do get transmitted back to the caller through parameter passing.
- Any changes to the formal parameter are reflected in the actual parameter in the calling environment as formal parameter receives a reference (or pointer) to the actual data



Object Oriented Analysis and Design with Java

Parameter Passing

Reference types continued..

- **Trick 1: The changes are not reflected back if we change the object itself to refer some other location or object**

```
class Test
{   int x;      Test(int i) { x = i; };     Test()      { x = 0; }
}
class Main2
{
    public static void main(String[] args)
    {   Test t = new Test(5);
        System.out.println(t.x);
        change(t);
        System.out.println(t.x);
    }
    public static void change(Test t)
    {   t = new Test();     t.x = 10;       }
}
```

- **Trick 2: Changes are reflected back if we do not assign reference to a new location or object**

```
class Test
{   int x;      Test(int i) { x = i; };     Test()      { x = 0; }
}
class Main2
{
    public static void main(String[] args)
    {   Test t = new Test(5);
        System.out.println(t.x);
        change(t);
        System.out.println(t.x);
    }
    public static void change(Test t)
    {    t.x = 10;  }
}
```

PES UNIVERSITY
CELEBRATING 50 YEARS

Coding examples – Demo

Parameter Passing

Acknowledgement : Prof Sindhu R Pai

THANK YOU

Prof.Priya Badarinath

Department of Computer Science and Engineering

priyab@pes.edu

Object Oriented Analysis and Design with Java - UE19CS353

Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

Overloading of Methods

Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

Agenda

1. Introduction

2. Coding examples - Demo

Object Oriented Analysis and Design with Java

Method Overloading

Introduction

- A feature that allows a class to have more than one method having the same name, if their argument lists are different.
- Method overloading is also known - Compile Time polymorphism, Static polymorphism , Early Binding.
- **Three ways to overload :** The argument lists of the methods must differ in either of these:
  - Changing the number of parameters
  - Changing the data type of parameters
  - Changing the order of parameters of methods.

Note: Method overloading has no relation with return-type

Method Overloading

Benefits of using Method Overloading

1. Method overloading increases the readability of the program.
2. This provides flexibility to programmers so that they can call the same method for different types of data.
3. This makes the code look clean.
4. This reduces the execution time because the binding is done in compilation time itself.
5. Method overloading minimises the complexity of the code.
6. The code can be used again, which saves memory.

Method Overloading

Coding Example 1: Number of parameters and Data type of parameters

```
class Bird{
    void fly(){
        System.out.println("Bird is flying");
    }
    void fly(int height){
        System.out.println("Bird is flying "+height+" high");
    }
    void fly(String name,int height){
        System.out.println(name+" is flying "+height+" feet high");
    }
}
class P2_methodOverload {
public static void main(String[] args){
        Bird bird1=new Bird();
        bird1.fly();
        bird1.fly(10000);
        bird1.fly("Eagle",10000);
    }
}
```

PES UNIVERSITY ONLINE

Coding Example 2: Number of parameters and Sequence of Data type of parameters

```
class Addition {
    int add(int a, int b) {
        return a + b; }
    int add(int a, int b, int c) {
        return a + b + c; }
    double add(int a, double b) {
        return a + b;    }
    double add(double a, int b) {
        return a + b;    }
}
class P2_MethodOverload {
    public static void main(String[] args) {
        Addition a = new Addition();
        int intAdd1 = a.add(1, 2);
        System.out.println("1+2=" + intAdd1);
        int intAdd2 = a.add(1, 2, 3);
        System.out.println("1+2+3=" + intAdd2);
        double doubleAdd1 = a.add(1, 2.5);
        System.out.println("1+2.5=" + doubleAdd1);
        double doubleAdd2=a.add(3.5,2);
        System.out.println("3.5+2=" + doubleAdd2);
    }   }
```

Object Oriented Analysis and Design with Java

Method Overloading

Acknowledgement - Prof. Sindhu R Pai

THANK YOU

Prof. Priya Badarinath
Department of Computer Science and Engineering

priyab@pes.edu

Object Oriented Analysis and Design using Java

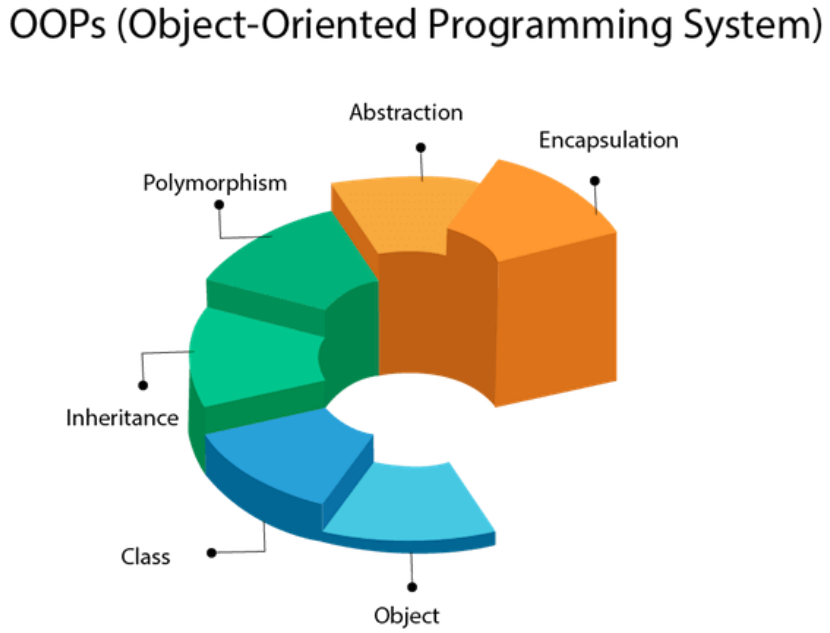
Prof: Mahitha G

Teaching Assistants : **Nishanth M S**
Vinay Padegal
Muskan Bansal

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Recursion



Mahitha G
Department of Computer Science and Engineering

T1 : Chapter 6: Introducing Classes

1: “Java the Complete Reference”, Herbert Schildt ,McGraw-Hill ,11th Edition, 2018.

Recursion is a process in which a method calls itself continuously.
A method in java that calls itself is called recursive method.

Recursion has two steps :

1. Recursive call on a problem of smaller size
2. Base cases or escape hatch for which the solution is directly available

PES UNIVERSITY

Recursion is used to solved many problems like:

Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc

One simple example.

How to find the sum of the digits of a given number?

Can we say that if the number is 0, then the sum is 0? Can we say that if the number has some digits, then the result is given by the unit digit + the sum of the digits of the number with that unit digit removed? If we keep removing the digits, ultimately the number has to become zero.

Sum of digits program

Example programs:

StarDraw.java

Factorial.java

THANK YOU

Mahitha G

Department of Computer Science and Engineering

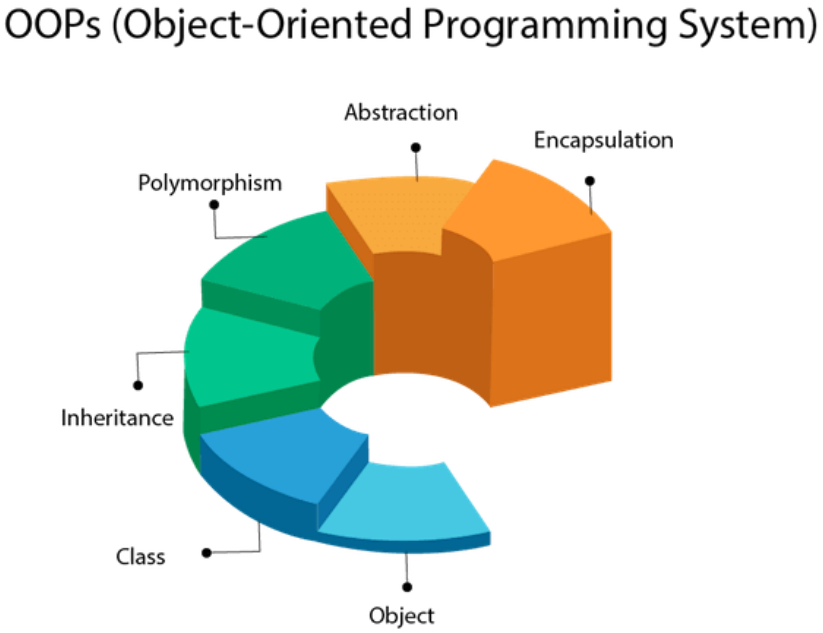
mahithag@pes.edu

Object Oriented Analysis and Design with Java (UE20CS352)

Prof: Mahitha G
Teaching Assistants – Vinay Padegal
Muskan Bansal
Nishanth M S
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Introduction to OO Programming



Mahitha G
Department of Computer Science and Engineering

PES UNIVERSITY
CELEBRATING 50 YEARS

T1 : Chapter 6: Introducing Classes

1: “Java the Complete Reference”, Herbert Schildt ,McGraw-Hill ,11th Edition, 2018.

Method Types

There are two types of methods:

Instance Methods

Instance method are methods which require an object of its class to be created before it can be called. To invoke a instance method, we have to create an Object of the class in which the method is defined.

Static Methods

Static methods are the methods in Java that can be called without creating an object of class.
They are referenced by the class name itself or reference to the Object of that class.

By default, methods are said to be instance methods

Method Types

Example to illustrate instance methods:

```
class Sample {

    String name = "“ ;

    public void  setname(String  s) { ame = s; }
}

class example1 {
    public static void main(String[] args)
    {

        // create an instance of the class.
        Sample ob1 = new Sample();

        // calling an instance method i
        ob1.setname(“ Ramu");
        System.out.println(ob1.name);
    }
}
```

Method Types

Example to illustrate static methods:

```
class Sample {

        public static String name = "";

        public static void setname(String s) {
                name= s;
        }
}

class example2 {
        public static void main(String[] args) {

                // Accessing the static method
setname)
                // and field by class name itself.
                Sample.setname("abhiram");
                System.out.println(Sample.name);


                // Accessing the static method
setname
                Sample obj = new Sample();
                System.out.println(obj.name);
                obj.setname("manish");
```

```
abhiram
abhiram
manish
```

Static Keyword

The static keyword can be used for :

- Variable (also known as a class variable)
- Method (also known as a class method)
- Block

Class variables (or static fields)

- Variables that are common to all objects
- They are associated with the class, rather than with any object
- Every instance of the class shares a class variable, which is in one fixed location in memory
- Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class

Static Keyword

Example to illustrate static variable:

TestStaticVariable.java

Example counter without static variable

Counter.java

Example counter with static variable

Counter2.java

Why Java main method is static????

Static Keyword

Java static block
Is used to initialize the static data member.
It is executed before the main method at the time of classloading.

Example of static block

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

Output:

```
static block is invoked
Hello main
```

Can we execute a program without main() method? Is static method enough???

static Block

- Static block may be used to initialize static variables
- Static block gets executed exactly once, when the class is first loaded

StaticBlockDemo.java

References

https://www.javatpoint.com/static-keyword-in-java

THANK YOU

Mahitha G (mahithag@pes.edu)
Teaching Assistants – Vinay Padegal
Muskan Bansal
Nishanth M S
Department of Computer Science and Engineering

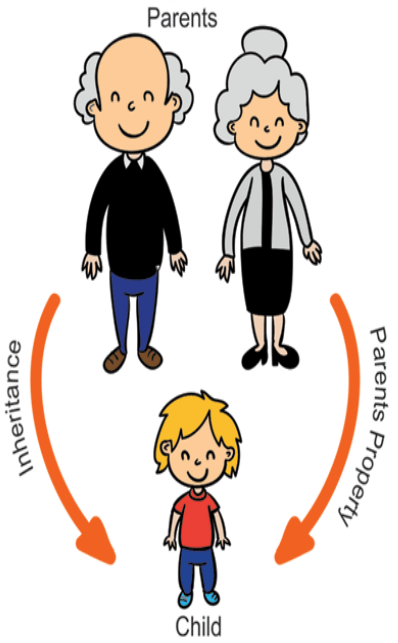
Object Oriented Analysis and Design with Java - UE20CS352

Mahitha G
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S

Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

Object Oriented Concepts

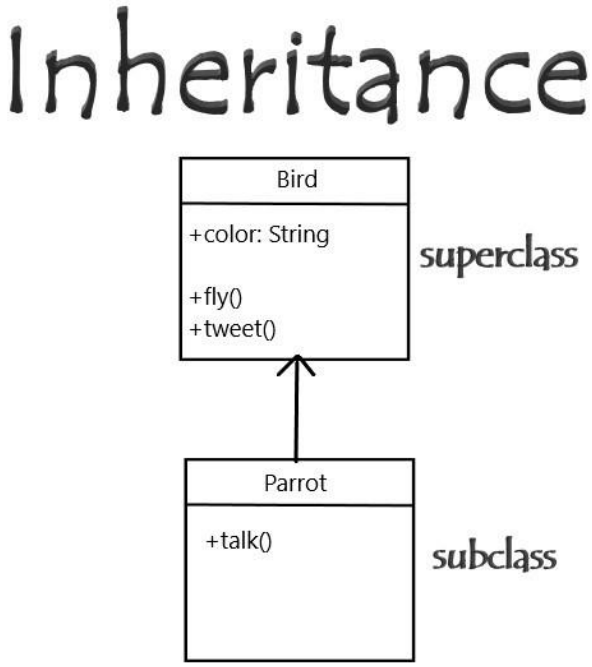


Mahitha G

Department of Computer Science and Engineering

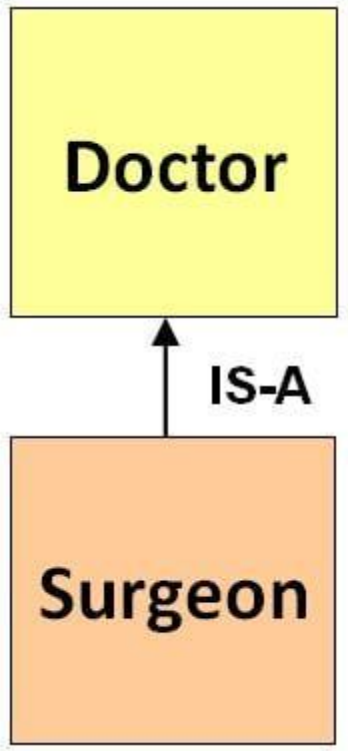
Inheritance

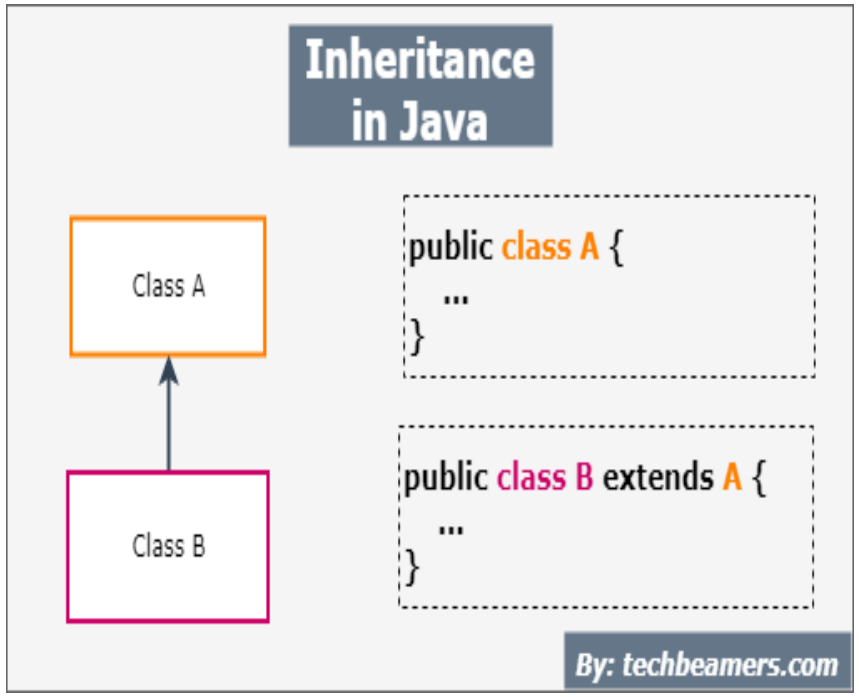
- Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- Inheritance represents IS-A relationship which is also known as a parent-child relationship
- Acquiring the properties (data and methods) from one class to other classes enables reusability of code.
- The class whose features are inherited is known as superclass (or a base class or a parent class), and the class to which its inherited to is called as subclass or child class

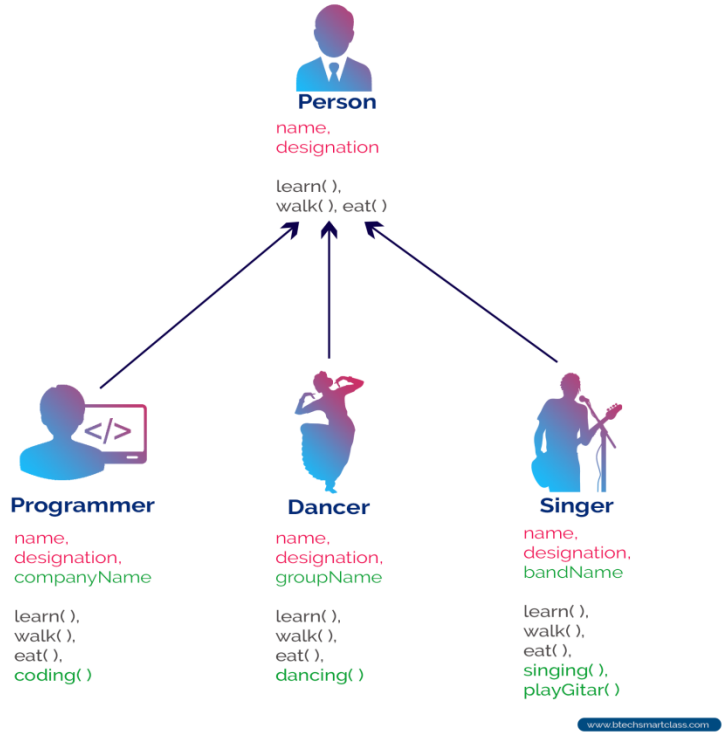


Inheritance

- The child class is a specific type of the parent class.







Inheritance

Syntax:

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

Object Oriented Analysis and Design with Java

Inheritance in Java

```java
class Bird {
    public String colour;
    public void fly(){ System.out.println("Bird is flying"); }
    public void tweet(){ System.out.println("Tweet tweet"); }
}

class Parrot extends Bird {
    public void fly(){ System.out.println("Parrot is flying"); }
    public void tweet(){ System.out.println("Parrot tweet"); }
    public void talk(){ System.out.println("Parrots can talk"); }
}

class Main {
    public static void main(String[] args) {
        Bird b = new Bird();
        Parrot p = new Parrot();
        p.colour = "Green";
        b.fly();
        b.tweet();
        p.fly();
        p.tweet();
        p.talk();

    }
}
```

```
Bird is flying
Tweet tweet
Parrot is flying
Parrot tweet
Parrots can talk
```

Inheritance in Java

```
class Teacher {
    String designation = "Teacher";
    String collegeName = "PESU";
    void does(){
        System.out.println("Teaching");
    }
}
public class JavaTeacher extends Teacher{
    String mainSubject = "Java";
    public static void main(String args[]){
        JavaTeacher obj = new JavaTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
    }
}
```

Method Over-riding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

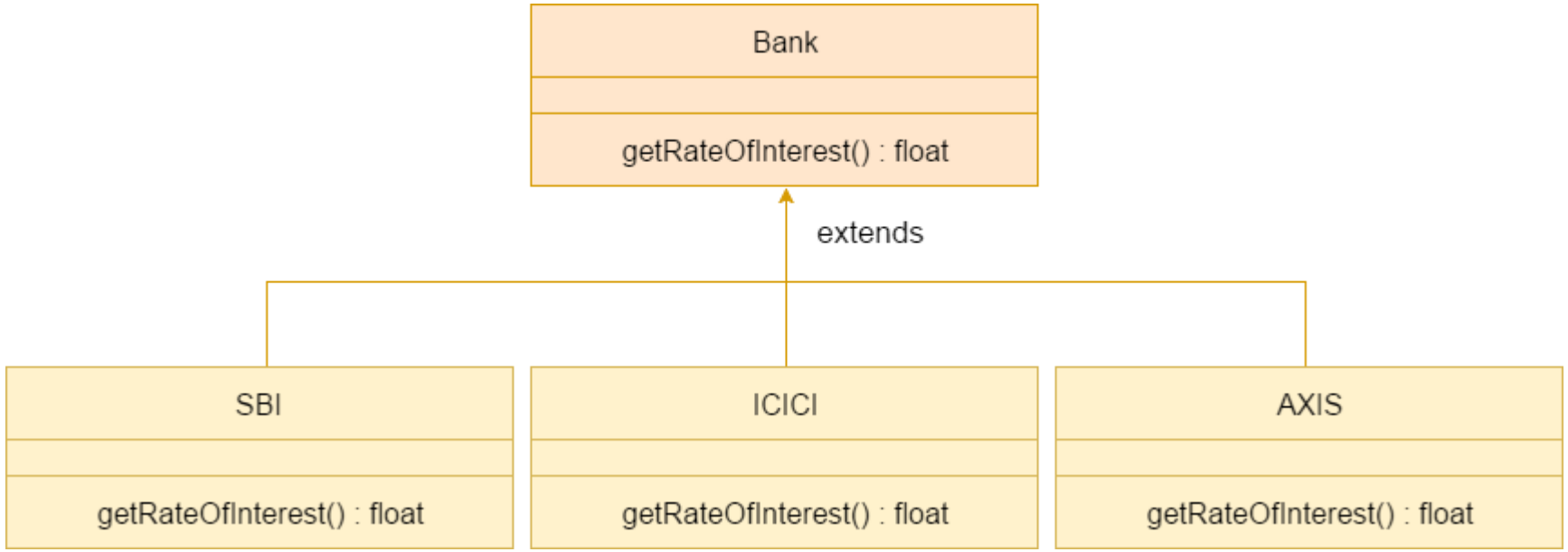
Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Method Over-riding

A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.

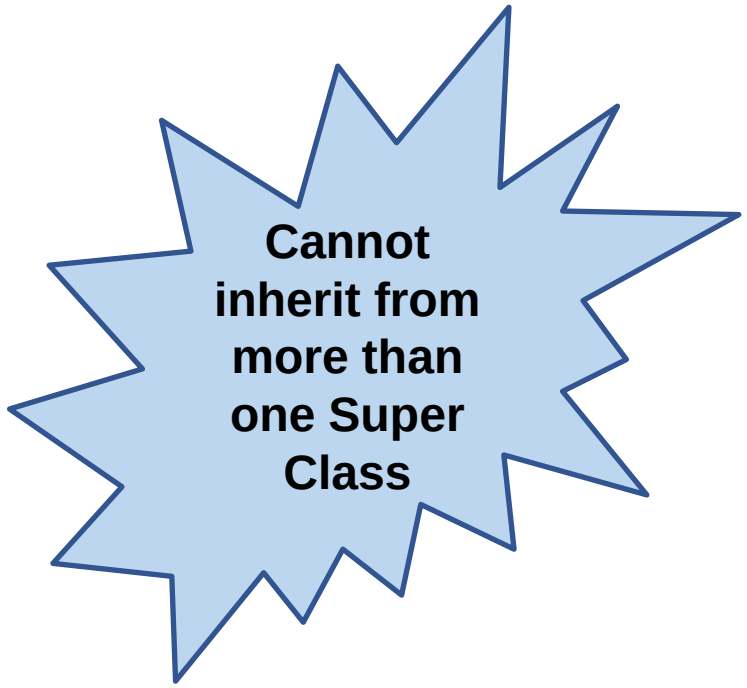


Inheritance in Java

```
class A {
    public void disp() { System.out.println("A: Display"); }
}

class B {
    public void disp() { System.out.println("B: Display"); }
}

class C extends A, B {
    public static void main(String[] args) {
        C c = new C();
        c.disp(); // which disp will it call?
    }
}
```



Why not??

Advantages:

- Reusability and saves time
- Enhances Readability
- Overriding

With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

When child class inherits the properties and functionality of parent class, we need not write the same code again in child class. This makes it easier to reuse the code, makes us write the less code and the code becomes much more readable and saves time.

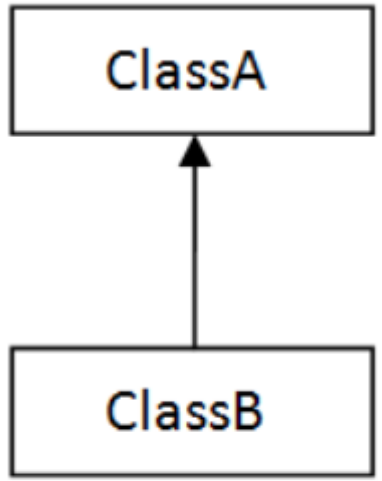
Types of Inheritance

- Single Level Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

Types of Inheritance -Example

Single Level Inheritance:

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

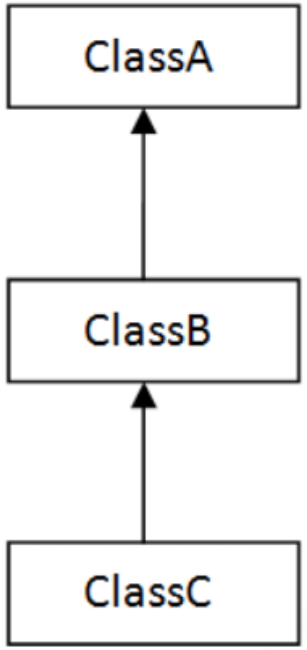


1) Single

Types of Inheritance

Multilevel Inheritance

```
class Animal{
      void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
      void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
      void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
   public static void main(String args[]){
      BabyDog d=new BabyDog();
      d.weep();
      d.bark();
      d.eat();
   }
}
```

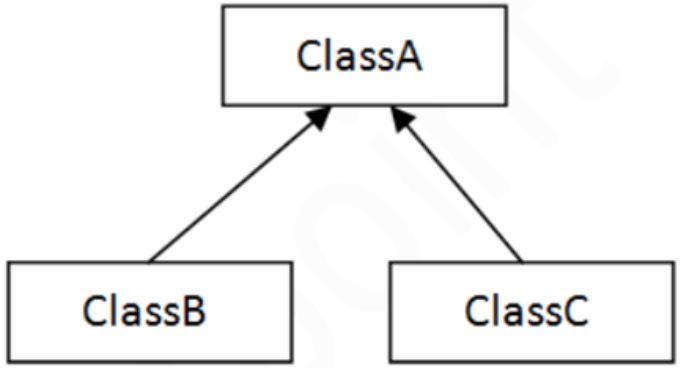


2) Multilevel

Types of Inheritance

Hierarchical Inheritance

```
class Animal{
      void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
      void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
      void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
      public static void main(String args[]){
            Cat c=new Cat();
            c.meow();
            c.eat();
            //c.bark();//C.T.Error
      }
}
```



3) Hierarchical

Object Oriented Analysis and Design with Java

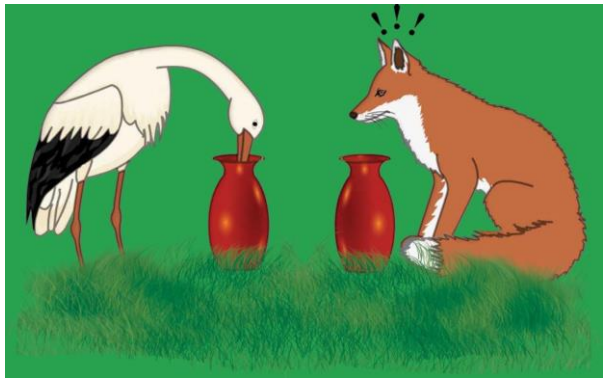
Subclass - Superclass Relationship

```
4  class Super {
5      void print() { System.out.println("Class super: print"); }
6  }
7
8  class Sub extends Super {
9      void print() { System.out.println("Class sub: print"); }
10     void show() { System.out.println("Class sub: show"); }
11 }
12
13 class Main {
14     public static void main(String[] args) {
15         Super obj1 = new Super();  // yes
16         obj1.print();  // ok
17
18         Sub obj2 = new Sub();  // yes
19         obj2.print();  // ok
20         obj2.show();  // ok
21
22         Super obj3 = new Sub();  // yes
23         obj3.print();  // ok
24         obj3.show();  // hidden, error
25
26         Sub obj4 = new Super();  // not allowed, error
27     }
28 }
```

Interfaces

You would have heard of the story of the fox and the stork – each one hosting a feast to the other. The fox serves the soup on a flat plate. The stork serves the soup in a pitcher with a narrow deep opening. The stork does not get the right interface to enjoy its meal when fox serves it. The fox does not get the right interface when the stork serves it. The moral of the story - interface matters the most.

So next topic is interfaces in java



Interfaces in java

•We know, objects define their interaction with the outside world through the methods that they expose.

•Methods form the object's interface with the outside world;

•Buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing.

•You press the "power" button to turn the television on and off.

In its most common form, an interface is a group of related methods with empty bodies.

Interfaces in java

An interface in Java says what the user of a class supporting this interface can do. It does not say how these will be implemented by the class supporting this interface. It says "what" and "not how".

- Interface defines how a certain type of object should behave
- What all should a certain type of object do to qualify as that type

It is possible that there could be a number of implementations for the same interface. The client can chose whatever he prefers – can also change from one to another. If you want to eat dosa, you can choose any eatery which supports the interface – Dosable – which makes dosa

Object Oriented Analysis and Design with Java

Interfaces in java

```
4  interface OpenableCloseable {
5      void open();
6      void close();
7  }
8
9  class NormalDoor implements OpenableCloseable {
10     public void open() { System.out.println("Pull to open"); }
11     public void close() { System.out.println("Push to close"); }
12 }
13
14 class SlidingDoor implements OpenableCloseable {
15     public void open() { System.out.println("Slide right to open"
           ); }
16     public void close() { System.out.println("Slide left to
           close"); }
17 }
```

Interface

A bicycle's behavior, if specified as an interface, might appear as follows:

```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

Interface

To implement the interface, the name of your class would change (to a particular brand of bicycle, for example, such as AAABicycle), and you would use the implements keyword in the class declaration:

```
class AAABicycle implements Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

   // The compiler will now require that methods
   // changeCadence, changeGear, speedUp, and applyBrakes
   // all be implemented. Compilation will fail if those
   // methods are missing from this class.

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

Why Interface

1. Used to achieve abstraction
2. Supports the functionality of Multiple inheritance
3. It can used to achieve loose coupling

Inheritance Vs Interface

Category	Inheritance	Interface
Description	Inheritance is the mechanism in java by which one class is allowed to inherit the features of another class.	Interface is the blueprint of the class. It specifies what a class must do and not how. Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
Use	It is used to get the features of another class.	It is used to provide total abstraction.
Syntax	class subclass_name extends superclass_name { }	interface <interface_name>{ }
Number of Inheritance	It is used to provide 4 types of inheritance. (multi-level, simple and hierarchical inheritance)	It is used to provide 1 types of inheritance (multiple).
Keywords	It uses extends keyword.	It uses implements keyword.
Inheritance	We can inherit lesser classes than Interface if we use Inheritance.	We can inherit enormously more classes than Inheritance, if we use Interface.
Method Definition	Methods can be defined inside the class in case of Inheritance.	Methods cannot be defined inside the class in case of Interface (except by using **static** and **default** keywords).
Multiple Inheritance	We cannot do multiple inheritance (causes compile time error).	We can do multiple inheritance using interfaces.

- **Can we instantiate an interface directly?**

NO. you cannot have a constructor. There is no default constructor. You can not make one either.

- **Can we have data members in an interface?**

We can. But these will for the whole class and will be immutable. In Java terminology, these will be static and final. So, the client of the class has a guarantee about these members. They exist in every class implementing the interface, can be accessed through the class or the object – no difference though – will never change

- **Can a class with all methods implemented also be abstract?**

It can be. If creating an object of that class does not make sense in the domain of application, the class can be made abstract.

Can we specify that the method of an interface is private?

◦ Definitely NO.

Can we specify that the method of an interface is protected?

◦ Interface should remain an interface for everybody in the world. The answer is a clear NO.

• **Can an interface extend an interface?** ◦

Definitely YES.

• **Can a class implement more than one interface?** ◦

No issue as there are no mutable members in the interface.

• **Can a class override a few methods of the interface which it implements?**

◦ Then the class remains abstract – therefore cannot be instantiated.

Object Oriented Analysis and Design with Java

Example Programs

Programs to demonstrate Inheritance and interface
Demo1.java
Demo2.java
Findarea.java //abs class + abs methods
Inherit.java
Interfacedemo.java

References

https://www.tutorialspoint.com/java/java_interfaces.htm#:~:text=An%20interface%20is%20a%20reference,abstract%20methods%20of%20the%20interface.

https://www.geeksforgeeks.org/are-all-methods-in-a-java-interface-are-abstract/

https://www.javatpoint.com/difference-between-abstract-class-and-interface

THANK YOU

Mahitha G (mahithag@pes.edu)
Teaching Assistants : Vinay Padegal
Muskan Bansal
Nishanth M S
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java - UE20CS352

Prof. Priya Badarinath & Prof. Sindhu Pai
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Abstract class and Object class

Prof. Priya Badarinath and Prof. Sindhu R Pai
Department of Computer Science and Engineering

Abstract class and Object class - Agenda

1. Introduction to Abstract class
2. Creation and Usage in Java
3. Coding examples – Demo
4. Introduction to Object class
5. Methods in Object class
6. Coding examples

Abstract class

Introduction

- Provides **implementation reuse – provides default implementation**
- To create a super class that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details.
- The super class determines the nature of methods that the subclasses must implement.
- Referred to as **subclasser responsibility** because they have no implementation specified in the superclass.
- No method body is present.

Abstract class

Creation and usage in Java

- Created using **abstract** keyword at the beginning of the class declaration.
- May contain abstract methods, i.e., methods without body
- If a class has **at least one abstract method**, then the **class must be declared abstract**
- **Cannot be instantiated**
- If a class extends abstract class then either it has to provide implementation of all abstract methods or declare this class as abstract class
- Can have both **static and non-static data members and methods** like any other java class
- **Can not be final** in Java because abstract classes are used only by extending
- A class **can extend only one abstract class** as Java does not support multiple inheritance

Abstract class

PES UNIVERSITY
CELEBRATING 50 YEARS

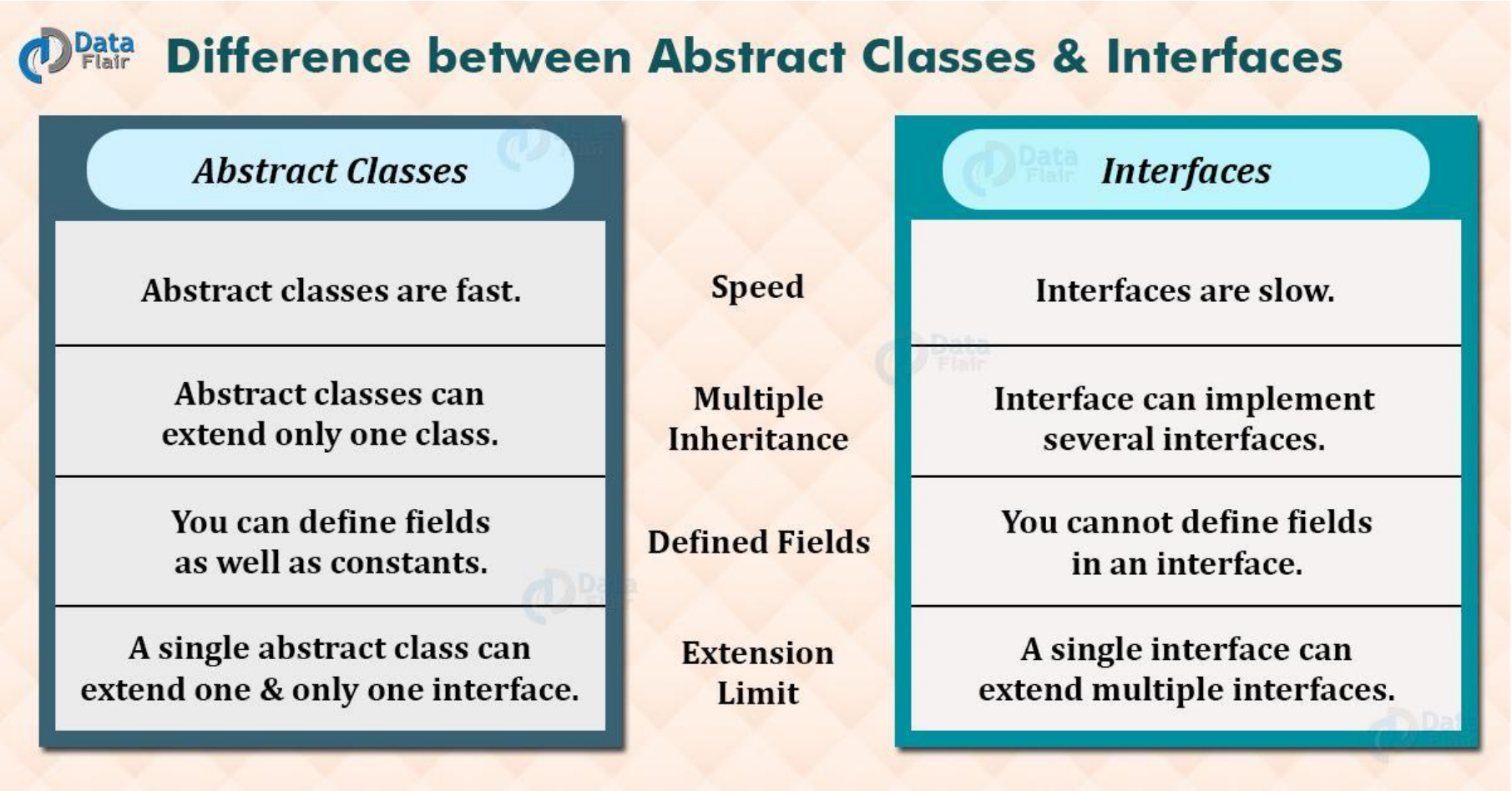
Coding Example: If the abstract class contains the below data, how to implement Rectangle and Triangle classes?

```
abstract class Figure {
    double dim1;
    double dim2;
    Figure(double a, double b) {
        dim1 = a;
        dim2 = b;
    }
    abstract double findArea();
}
```

class Rectangle extends Figure {
??
}

class Triangle extends Figure {
? ?
}

Abstraction Vs Interface



Single Rooted Hierarchy

- Same Base Class
- Common Interface
- It enables easy memory management
- Simplifies argument passing amongst object too on the heap.

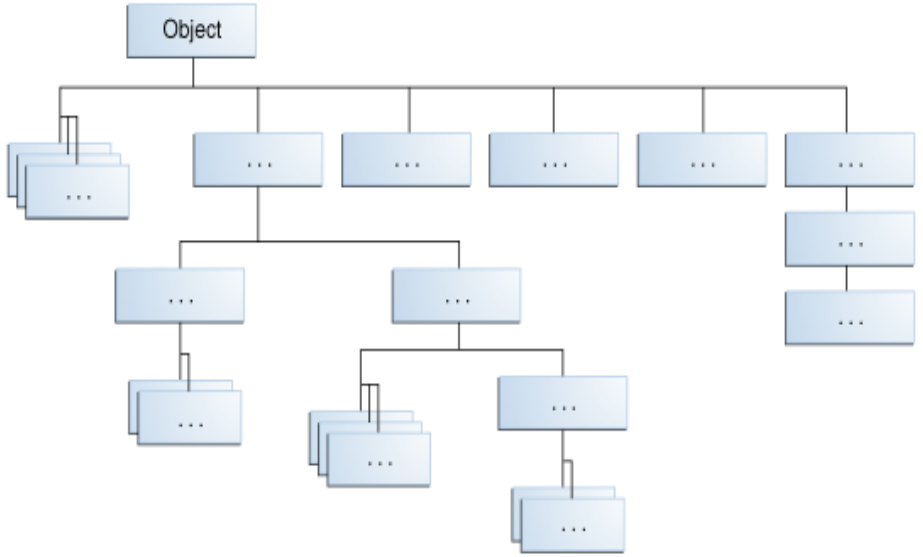
The singly-rooted hierarchy is common with most other object-oriented programming languages.

As Java was created from scratch, it has no backward compatibility issues with any existing language

Object class

Introduction

- Object class defined by Java is a super class of all other classes, in the absence of any other explicit superclass
- A reference variable of type Object can refer to an object of any class
- This is defined in the **java.lang** package



Object class

Methods: Object class defines some methods, which are available in every object

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

Coding example – 1: Demo of overriding toString() function

```
class Box {
    int width;
    int height;
    int depth;
    Box()
    {   this.width = 0; this.height = 0; this.depth = 0;
    }
    Box(int l,int m,int n)
    {   this.width = l; this.height = m; this.depth = n;
    }
    @Override
    public String toString() {
        return width + " " + height + " " + depth;
    }
}
```

```
public class P1_object {
    public static void main(String[] args) {
        Box obj = new Box();
        System.out.println(obj);
        Box new_obj = new Box(3,2,1);
        System.out.println(new_obj);
    }
}
```

Coding example – 2: Demo of overriding of equals() function

```java
class Box {
    int width;
    int height;
    int depth;
    Box()
    {   this.width = 0; this.height = 0; this.depth = 0;
    }
    Box(int l,int m,int n)
    {   this.width = l; this.height = m; this.depth = n;
    }
    @Override
    public boolean equals(Object o) {
        Box b2 = (Box) o; // imp
        return this.width == b2.width && this.height == b2.height && this.depth == b2.depth;
    }
}
```

```java
public class P2_Object {
    public static void main(String[] args) {
        Box obj1 = new Box();
        Box obj2 = new Box(3,2,1);
        Box obj3 = new Box(3,2,1);
        System.out.println(obj1 == obj2);
        System.out.println(obj2.equals(obj3));
    }
}
```

THANK YOU

Prof. Priya Badarinath and Sindhu R Pai
Department of Computer Science and Engineering

priyab@pes.edu, **sindhurpai@pes.edu**

Object Oriented Analysis and Design using Java - UE20CS352

Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Array, List and Stack

Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Abstract class and Object class - Agenda

1. Introduction to Array class
2. Creation and Usage of Array in Java
3. Coding examples – Demo
4. Introduction to List class
5. Creation and Usage of List interface in Java
6. Coding examples – Demo
7. Introduction to Stack class
8. Creation and Usage of Stack in Java
9. Coding examples – Demo

Introduction

- An Object of similar elements stored in contiguous memory allocation.
- Can be of any type
- Types: One Dimensional, Two Dimensional
- Declaration of 1D Array & Instantiation:
  - dataType[ ] arr; (or)
  - dataType [ ]arr; (or)
  - dataType arr[ ];

Instantiation:

arrayRefVar=**new** datatype[size];

- Declaration of 2D Array & Instantiation: :
  - dataType[ ][ ] arr; (or)
  - dataType [ ][ ]arr; (or)
  - dataType arr [ ][ ];

Instantiation:

int[][] arr=**new int**[3][3]; //3 rows and 3 columns

Array Representation

Representation of 1 Dimensional Array

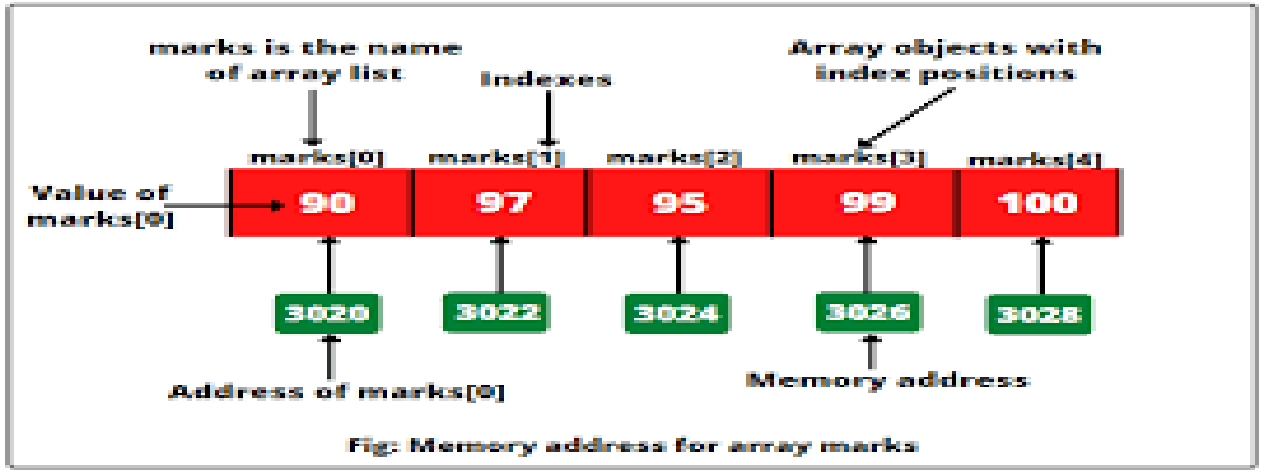


Fig: Memory address for array marks

Representation of 2 Dimensional Array

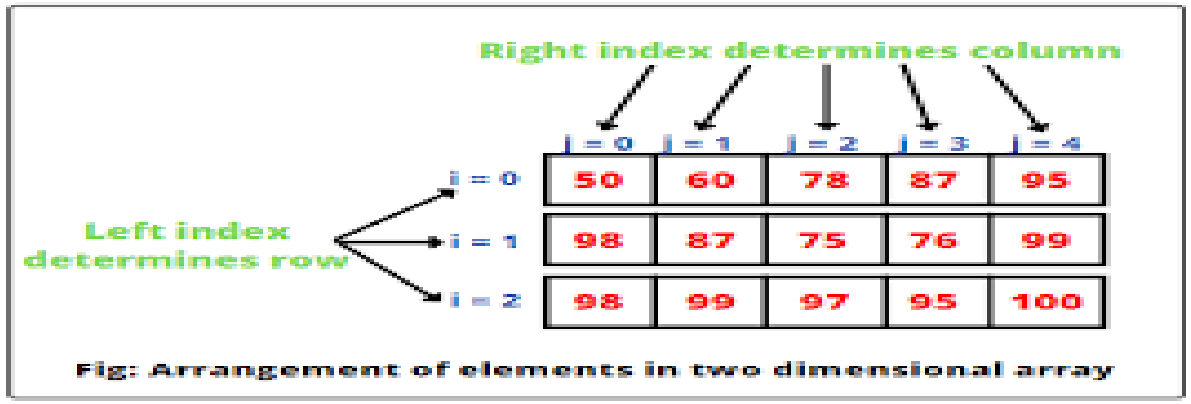


Fig: Arrangement of elements in two dimensional array

Foreach Loop

- Foreach loop prints the array elements one by one.
- It holds an array element in a variable, then executes the body of the loop.
- The syntax of the for-each loop is given below:
  - **for**(data_type variable:array){ …}.
- Ex : Coding Demo

Object Oriented Analysis and Design using Java

Arrays Class in Java

- Arrays class in java.util package is a part of the Java Collection Framework.
- It provides static methods to dynamically create and access Java arrays.
- It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.
- The class hierarchy is as follows:
    - java.lang.Object
    - ↳ java.util.Arrays.

Usage :

- Syntax: Class declaration
    - public class Arrays extends Object
- Syntax: In order to use Arrays : Arrays.<function name>;

Functions:

- binarySearch(), asList(),toString(),equals(),sort().

Object Oriented Analysis and Design using Java

Introduction to List Interface in Java

- *List* in Java provides the facility to maintain the *ordered collection.*
- It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.
- List interface is found in the java.util package and inherits the Collection interface.
- The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector.

Declarations: public interface List<E> extends Collection<E>

Few Methods :

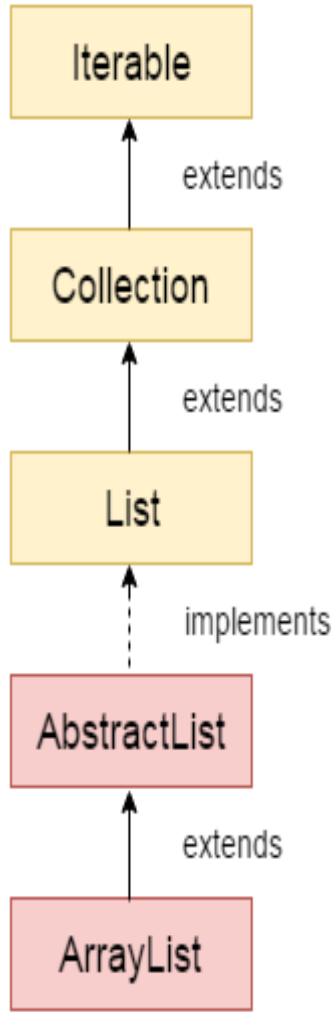
- size(), clear(), add(), Add(),addAll(), contains(), containsAll(), equals(), hashCode(), isEmpty(), indexOf(). etc...

Example : Programming Demo

Introduction to ArrayList Class in Java

- **ArrayList** class uses a ***dynamic array*** for storing the elements.
- No size limit.
- Can add or remove elements anytime.
- Found in java.util package.
- ArrayList in Java can have the duplicate elements also.
- It implements the List interface, all the methods of the List interface can be used.
- Java ArrayList class maintains insertion order.
- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases.
  - **For example:**

1. ArrayList<int> al = ArrayList<int>(); // does not work
2. ArrayList<Integer> al = new ArrayList<Integer>(); // works fine



Introduction to ArrayList Class in Java

Declaration :

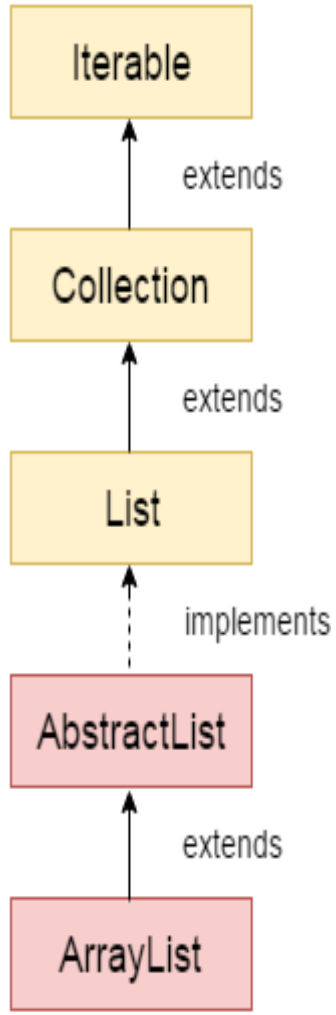
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable

Example for creating generic ArrayList:

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

Few Methods()

add(), addAll(), clear(), clone(), contains(), remove(), removeAll(), replace(), replaceAll(), etc…



Object Oriented Analysis and Design using Java

Introduction to Stack Class in Java

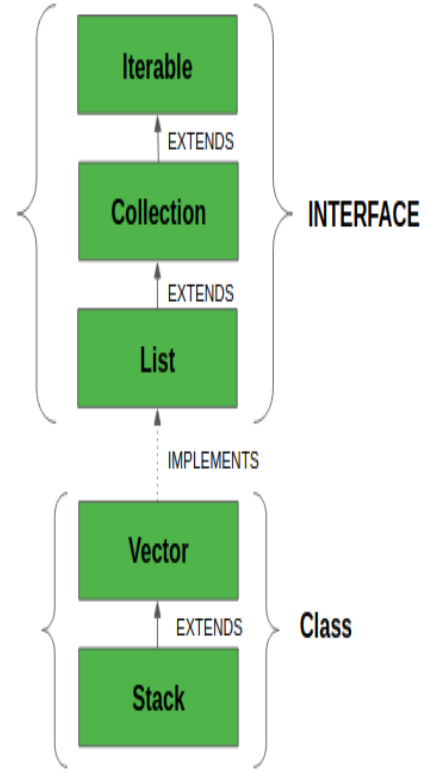
- Java Collection framework provides a Stack class that models and implements a **Stack data structure**.
- The class is based on the basic principle of last-in-first-out [LIFO].

Declaration:

public class Stack<E> extends Vector<E>

Implemented interfaces :

- **Serializable:** It is a marker interface that classes must implement if they are to be serialized and deserialized.
- **Cloneable:** This is an interface in Java which needs to be implemented by a class to allow its objects to be cloned.
- **Iterable<E>:** This interface represents a collection of objects which is iterable — meaning which can be iterated.
- **Collection<E>:** A Collection represents a group of objects known as its elements. The Collection interface is used to pass around collections of objects where maximum generality is desired.
- **List<E>:** The List interface provides a way to store the ordered collection. It is a child interface of Collection.
- **RandomAccess:** This is a marker interface used by List implementations to indicate that they support fast (generally constant time) random access.



Introduction to Stack Class in Java

- **creation of a stack class**
  - import **java.util.stack** package and use the Stack() constructor.

- **Few Methods:**
  - empty()
  - push(E item).
  - pop()
  - peek()
  - search(Object o)

- **Example: Program Demo**

THANK YOU

Prof. Priya Badarinath and Sindhu R Pai
Department of Computer Science and Engineering

priyab@pes.edu, **sindhurpai@pes.edu**