Object Oriented Analysis and Design Using Java

UE23CS352B

Unit:2

Object-oriented Programming: JVM

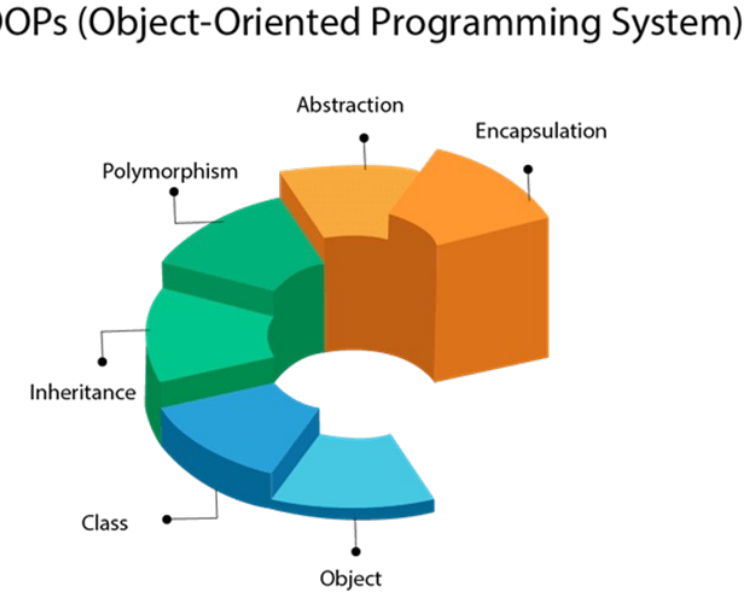
Prof . Shridevi A Sawant
Department of Computer Science and Engineering

UE23CS352B: Object Oriented Analysis and Design Using Java

Introduction to Object Oriented Programming

Prof . Shridevi A Sawant

Department of Computer Science and Engineering



Acknowledgement:
Prof . Shilpa S
Department of Computer Science and

Agenda

- Introduction to Java
- Features
- JVM
- JRE
- Java First Program

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)
- In 2010, Oracle Corporation acquired Sun Microsystems and became the steward of Java.

- 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

Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming languages
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has huge community support (tens of millions of developers)
- Java is an object-oriented language that gives a clear structure to programs and allows code to be reused, lowering development costs

Applications:

Mobile applications (specially Android apps), Desktop applications, Web applications

Web servers and application servers, Games, Database connection, And much, much more!

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.

Introduction to Java: 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.

6. Portable

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

7. Interpreted

The 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

Introduction to Java: Features

Common terms of Java:

- **Java Virtual Machine(JVM):** This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program. In the Running phase of a program, JVM executes the bytecode generated by the compiler.

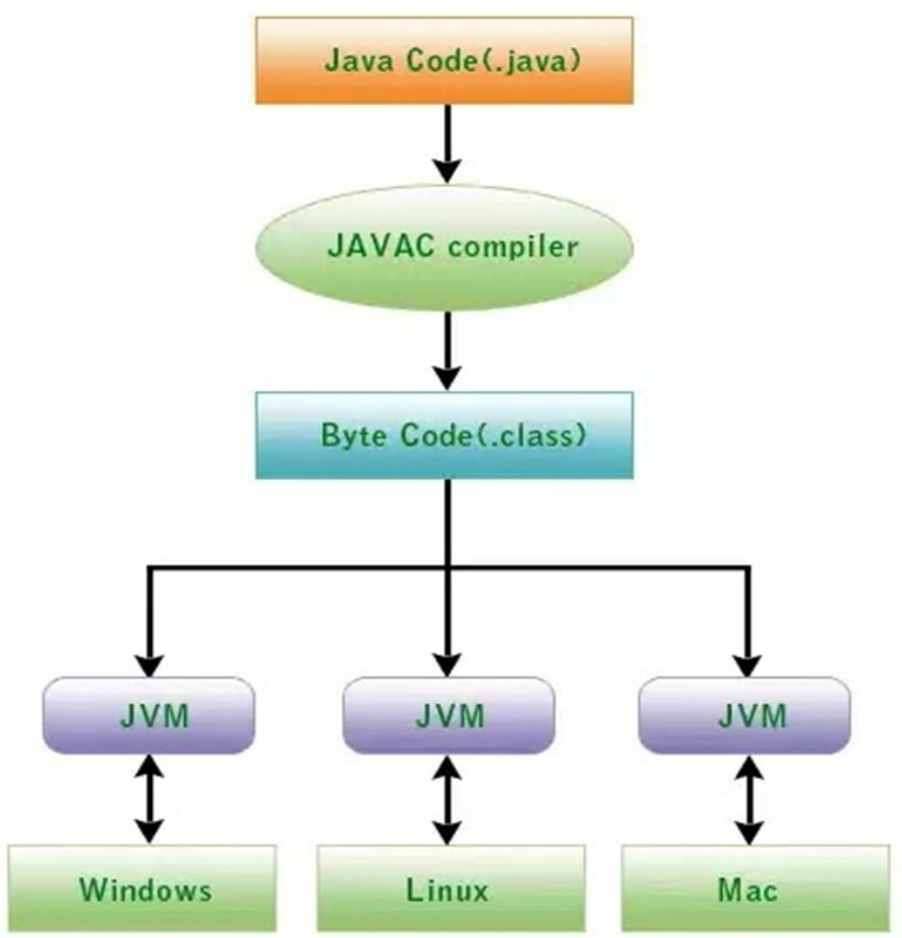
- **Java Development Kit(JDK):** It is a complete Java development kit that includes compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc.

- **Java Runtime Environment (JRE):** JDK includes JRE. JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet support, and plugins. For running the java program, a computer needs JRE.

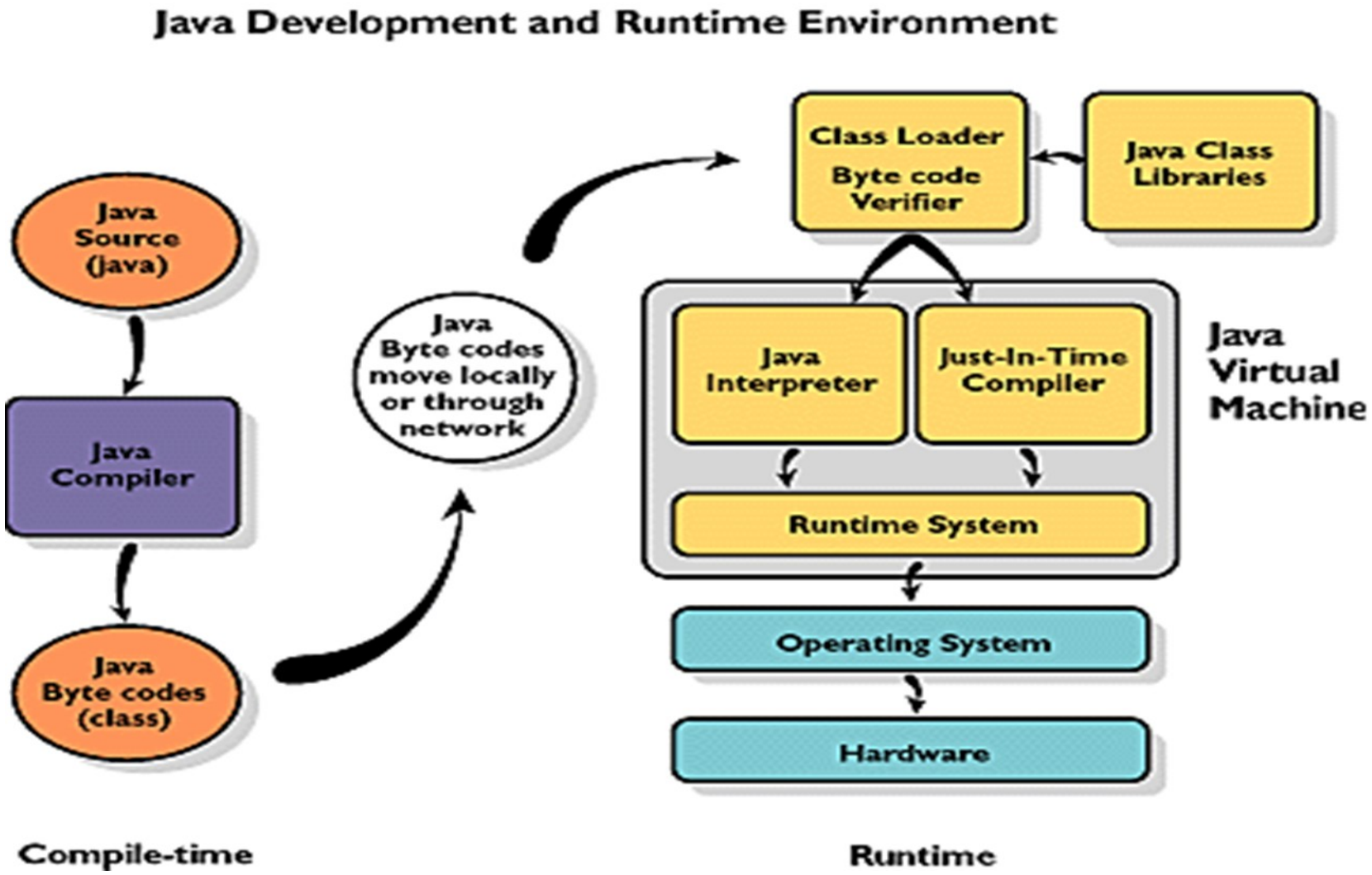
JAVA Virtual Machine

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

- **A specification** where the working of Java Virtual Machine is specified. But implementation provider is independent in choosing 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.



Java Runtime Environment(JRE)



Java First Program

Implementation of a Java application program involves the following steps.

1. Creating the program
2. Compiling the program
3. Running the program

Example:

```
  public class Test {
   public static void main(String[] args) {
       System.out.println("My Java First Program");
   }
}
```

Save file as -> Test.java
Compile -> javac Test.java
Run -> java Test

Output:

My Java First Program

PES UNIVERSITY

MCQ

1. **Java was mainly developed to create software for:**
   A. Scientific simulations
   B. Consumer electronic devices
   C. Operating systems
   D. Database servers

2. **Which Java component includes the compiler, debugger, and documentation tools?**
   A. JVM
   B. JRE
   C. JDK
   D. Bytecode

3. **The file generated after compiling a Java program has the extension:**
   A. .java
   B. .exe
   C. .class
   D. .obj

4. **The implementation of JVM is commonly known as:**
   A. JDK
   B. JRE
   C. Compiler
   D. Bytecode

5. Which command is used to compile Test.java?

A. java Test
B. javac Test
C. javac Test.java
D. compile Test.java

MCQ SOLUTIONS

1. B
2. C
3. C
4. B
5. C

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Engineering
shridevias@pes.edu

Object Oriented Analysis and Design Using Java

UE23CS352B

Unit:2

Abstraction, Encapsulation, Composition

Prof . Shridevi A Sawant
Department of Computer Science and Engineering

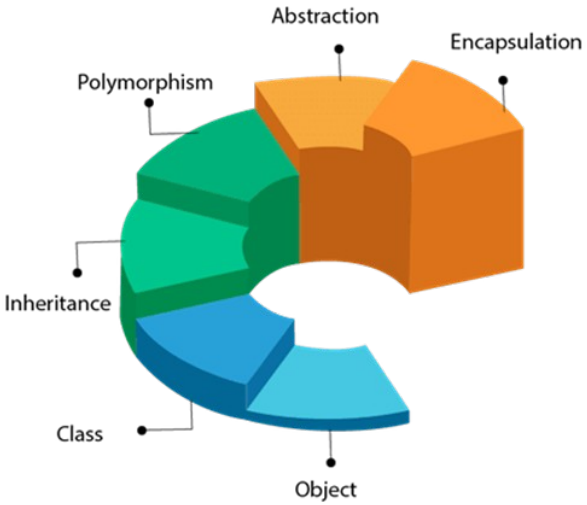
Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission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

UE23CS352B: Object Oriented Analysis and Design Using Java

Abstraction, Encapsulation, Composition

Prof . Shridevi A Sawant

Department of Computer Science and Engineering



Acknowledgements:
Prof. Mahitha G
Prof . Shilpa S
Department of Computer Science and Engineering

Agenda

- Abstraction
- Encapsulation
- Composition

Abstraction

Data abstraction in Object-oriented programming is a process of providing functionality to the users by hiding its implementation details from them. In other words, the user will have just the knowledge of what an entity is doing instead of its internal working.

In Java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.

Advantages of Java Abstraction:

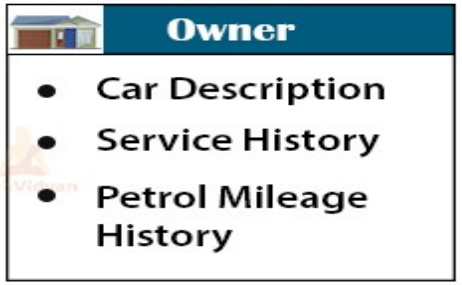
- Reduce Complexity
- Avoid code duplication
- Eases the burden of maintenance
- Increase Security and Confidentiality

Abstraction

Abstraction defines an object in terms of its properties (attributes), behavior (methods), and interfaces (means of communicating with other objects). 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 (A**

Example:



Abstraction

Abstract class in Java:

- In Java, we can achieve Data Abstraction using Abstract classes and interfaces.
- Interfaces allow 100% abstraction (complete abstraction).
- An Abstract class is a class whose objects can’t be created.
- An Abstract class is created through the use of the abstract keyword. It is used to represent a concept.
- An abstract class can have abstract methods (methods without body) as well as non-abstract methods or concrete methods (methods with the body). A non-abstract class cannot have abstract methods.

Abstraction

Abstract class in Java:

- The class has to be declared as abstract if it contains at least one abstract method.
- An abstract class does not allow you to create objects of its type. In this case, we can only use the objects of its subclass.
- Using an abstract class, we can achieve 0 to 100% abstraction.
- There is always a default constructor in an abstract class, it can also have a parameterized constructor.
- The abstract class can also contain final and static methods.

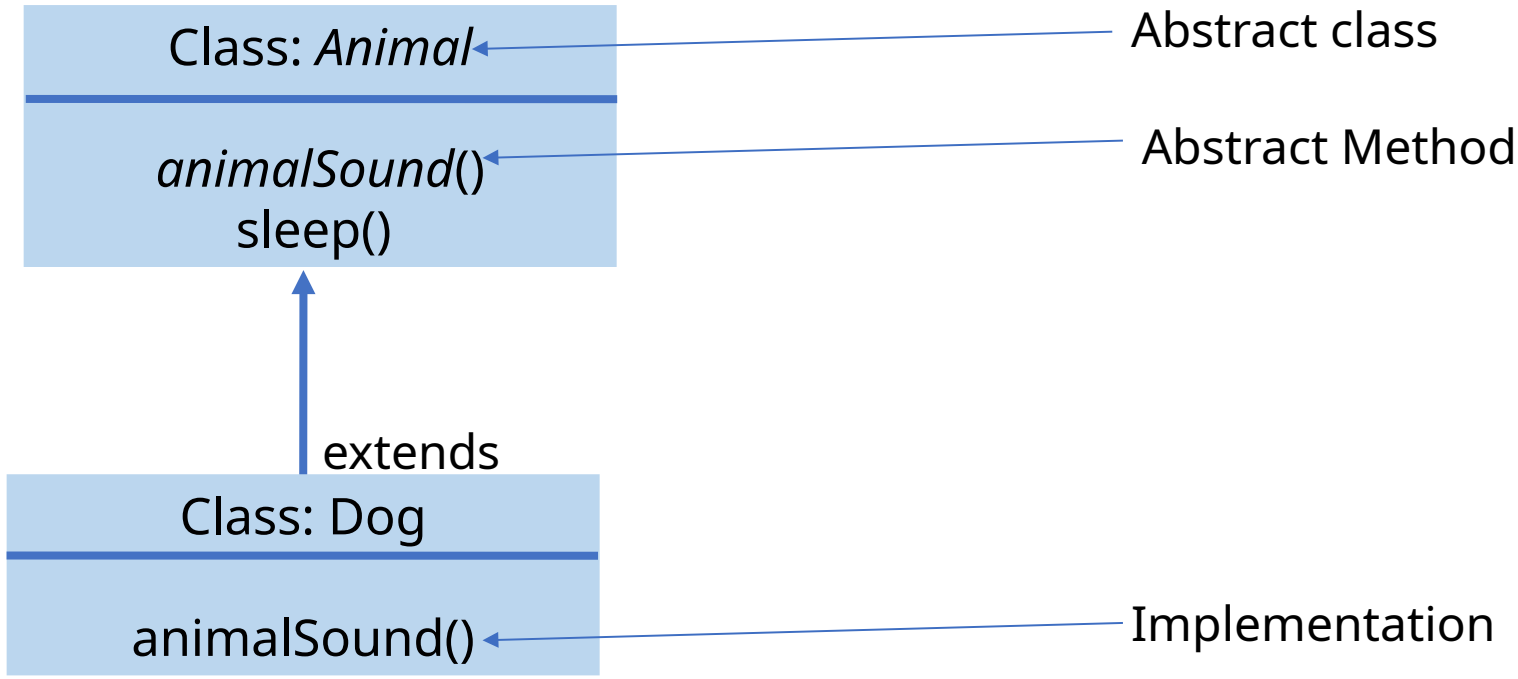
Abstraction : Example

```
abstract class Animal {   // Abstract class

    public abstract void animalSound();  // Abstract method

    public void sleep() {              // Regular method
        System.out.println("Zzz");
    }
}

class Dog extends Animal {             // Subclass

    @Override
    public void animalSound() {
        System.out.println("The dog says: bow bow");
    }
}

class Main {

    public static void main(String[] args) {
        Dog myDog = new Dog();          // Create a Dog object
        myDog.animalSound();
        myDog.sleep();
    }
}
```

Output:

The Dog says: bow bow
Zzz

Abstraction : Example



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.

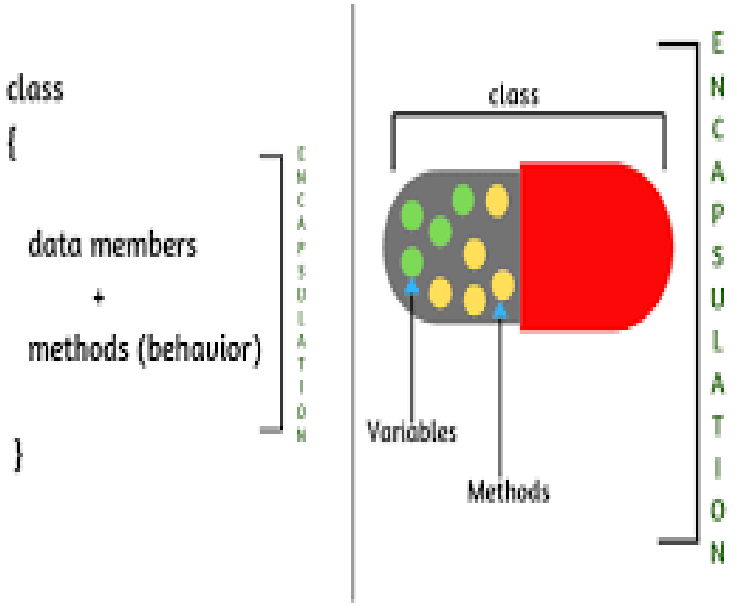


Fig: Encapsulation

Encapsulation

Advantages of Encapsulation

- **Data Hiding**: it is a way of restricting the access of our data members by hiding the implementation details.
- **Increased Flexibility**: We can make the variables of the class read-only or write-only depending on our requirements.
- **Reusability**: Encapsulation also improves the re-usability and is easy to change with new requirements.
- **Testing code is easy**: Encapsulated code is easy to test for unit testing.
- **Freedom to the programmer in implementing the details of the system**: This is one of the major advantages of encapsulation that it gives the programmer freedom in implementing the details of a system.

Encapsulation

Disadvantages of Encapsulation in Java

- Can lead to increased complexity, especially if not used properly.
- Can make it more difficult to understand how the system works.
- May limit the flexibility of the implementation.

Encapsulation: Example

```
class Person {

    private String name;   // private = restricted access

    public String getName() {    // Getter
        return name;
    }

    public void setName(String newName) {    // Setter
        this.name = newName;
    }
}

class Main {

    public static void main(String[] args) {
        Person myObj = new Person();

        myObj.name = "John";                // error
        System.out.println(myObj.name);      // error
    }
}
```

Output:

ERROR!

javac/tmp/mNvaXCz9rO/Main.java/tmp/mNvaXCz9rO/Main.java:15: error: name has private access in Person

myObj.name = "John";   //error

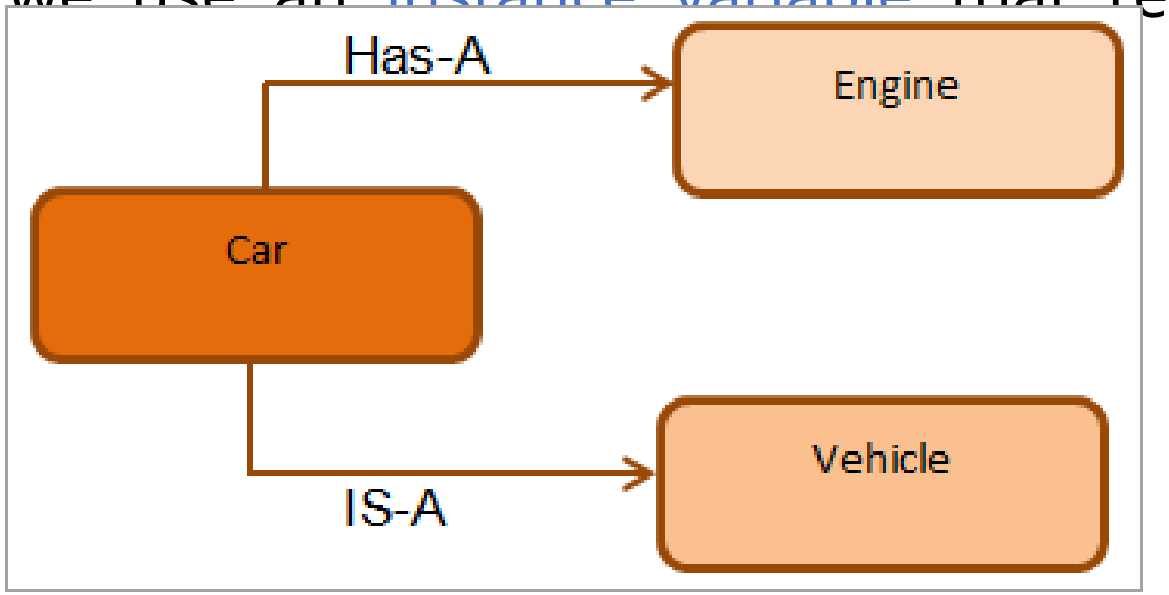
/tmp/mNvaXCz9rO/Main.java:16: error: name has private access in Person

System.out.println(myObj.name); // error

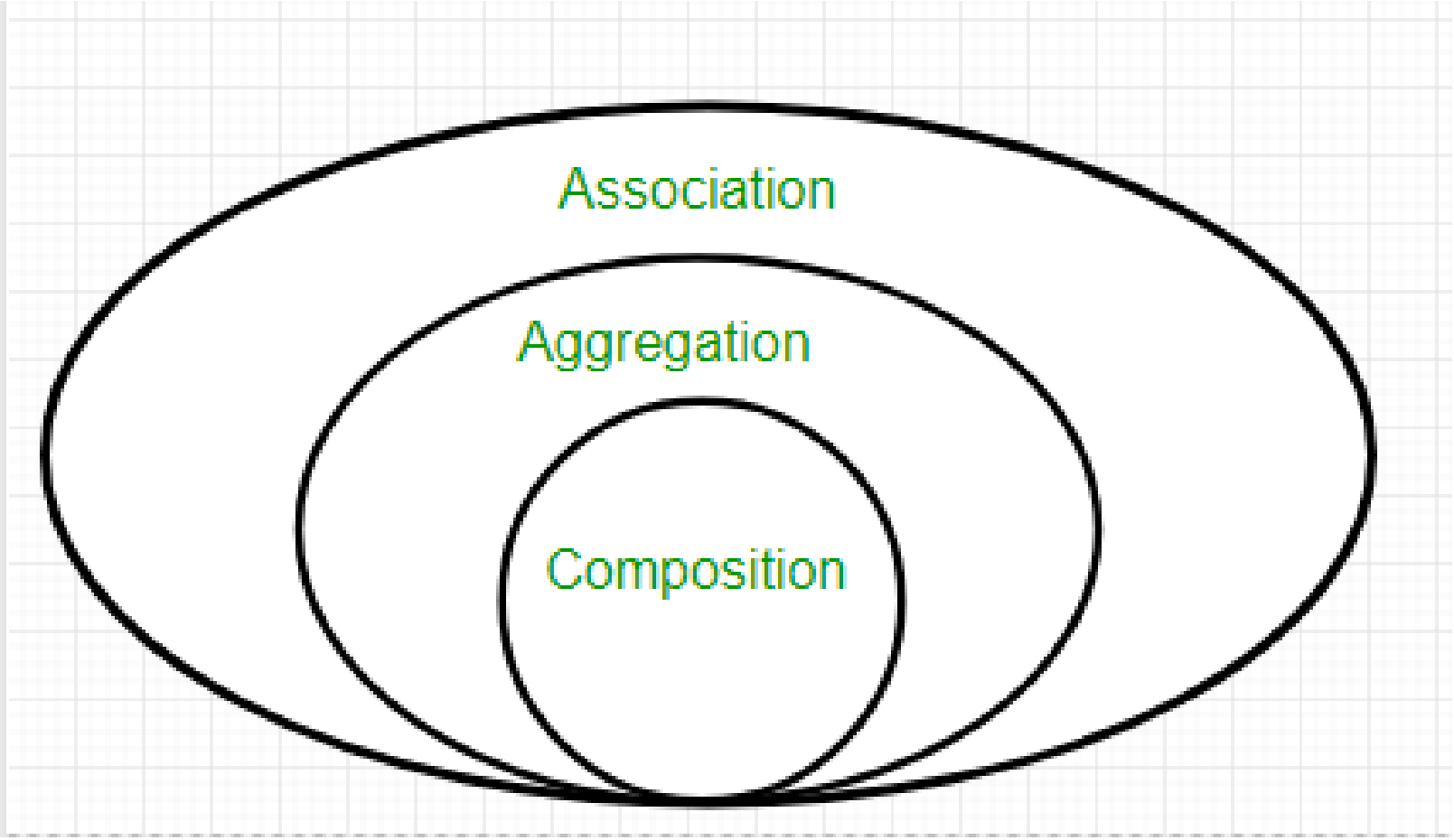
^2 errors

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.



Composition



Composition

- The Composition represents a part-of relationship.
- Both entities are related to each other in the Composition.
- The Composition between two entities is done when an object contains a composed object, and the composed object cannot exist without another entity. For example, if a university HAS-A college-lists, then a university is a whole, and college-lists are parts of that university.
- Favor Composition over Inheritance.
- If a university is deleted, then all corresponding colleges for that university should be deleted.

Composition

Benefits of using Composition:

- Composition allows us to reuse the code.
- In Java, we can use multiple Inheritance by using the composition concept.
- The Composition provides better test-ability of a class.
- Composition allows us to easily replace the composed class implementation with a better and improved version.
- Composition allows us to dynamically change our program's behavior by changing the member objects at run time.

Composition : Example

```
class Engine {
    void start() {
        System.out.println("Engine started");
    }
}

class Car {
    private Engine engine = new Engine(); // Engine is created inside Car → strong ownership

    void drive() {
        engine.start();
        System.out.println("Car is moving");
    }
}
```

Composition : Example

```
public class Main {

    public static void main(String[] args) {
        Car car = new Car();
        car.drive();
    }

}
```

OUTPUT

Engine started
Car is moving

MCQ

1. **Data abstraction in object-oriented programming focuses on:**
   A. Showing internal implementation details
   B. Representing only background details
   C. Providing functionality while hiding implementation details
   D. Eliminating methods from classes
2. **In Java, 100% abstraction can be achieved using:**
   A. Abstract classes only
   B. Concrete classes
   C. Interfaces
   D. Final classes

MCQ

3. Encapsulation in object-oriented programming refers to:

A. Separating data and methods
B. Wrapping data and functions into a single unit
C. Exposing internal data to the user
D. Eliminating data members

4. In encapsulation, data members of a class are accessed:

A. Directly by the program
B. Only through wrapped functions (methods)
C. By any external class
D. Without any restriction

MCQ

5. Composition in object-oriented programming represents a:

A. Has-a relationship
B. Is-a relationship
C. Part-of relationship
D. Uses-a relationship

6. Which benefit of composition allows behavior to change at runtime?

A. Code reuse
B. Static binding
C. Dynamic replacement of member objects
D. Compile-time polymorphism

MCQ SOLUTION

1. C
2. C
3. B
4. B
5. A
6. C

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Engineering
shridevias@pes.edu

Object Oriented Analysis and Design (UE23CS352B)

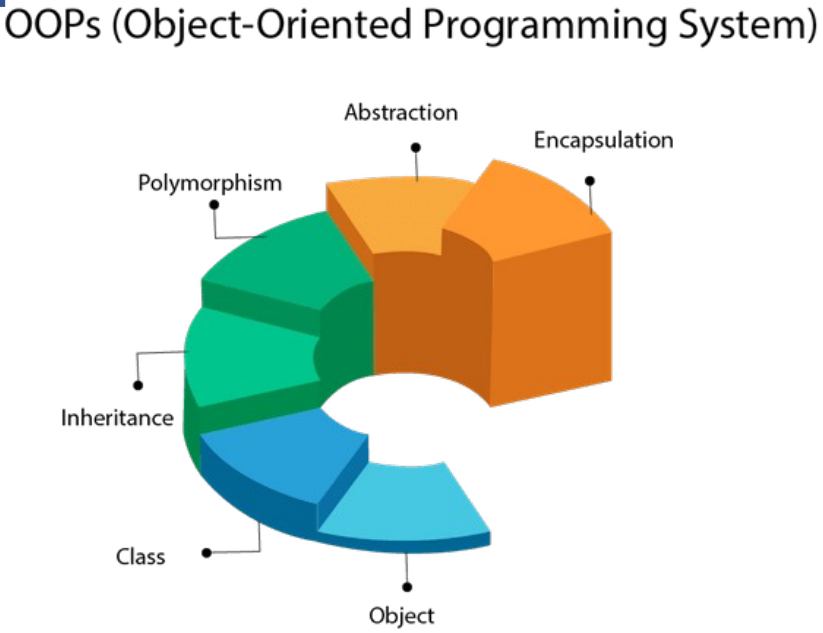
Prof Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design using java

Class, Attributes, Methods and Objects

Prof. Shridevi A Sawant

Department of Computer Science and Engineering



What is a Class in Java?

A class is a blueprint or template used to create objects.

It defines what an object will have (**attributes**) and what an object can do (**methods**).

Think of a class like a design of a house, and objects are the actual houses built from it.

Object Oriented Analysis and Design using Java

Object Oriented Programming: Structure of a Class

```
class ClassName {
    // attributes (variables)
    // methods (functions)
}
```

Object Oriented Analysis and Design using Java

Class : Attributes

- Attributes represent the properties or data of an object.
- They store information about the object.
- Hence they are called data members of the class.

```
class Student {
    int rollNo;
    String name;
}
```

rollNo → attribute
name → attribute

Object Oriented Analysis and Design using Java

Class : Methods

- **Methods** define the **behavior or actions** an object can perform.
- It **uses attributes** to perform an action

```
class Student {
    int rollNo;
    String name;

    void display() {
        System.out.println(rollNo + " " + name);
    }
}
```

rollNo → attribute
name → attribute

display() → method

Class - Example

```
class Car {
    // Attributes
    String brand;
    int speed;

    // Method
    void drive() {
        System.out.println("Car is driving at " + speed + " km/h");
    }
}
```

Java: Object

What is an Object in Java?

- An **object** is a real-world entity **created from a class**.
- It represents a specific **instance of a class** and occupies memory.
- If a class is a blueprint, an object is the actual thing made from it.

Object: An instance of a class that contains **state (attributes)** and **behavior (methods)**.

- Attributes and methods can be accessed via dot(.) operator.
- Object (instance) of a class is created with '**new**' keyword.

Object Oriented Analysis and Design using Java

Class and Objects

```
class Student {
    int rollNo;
    String name;

    void display() {
        System.out.println(rollNo + " " + name);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();  // object

        s1.rollNo = 1;
        s1.name = "John";

        s1.display();
    }
}
```

An object has three things:

Identity – Unique reference (s1)

State – Values of attributes (rollNo, name)

Behavior – Methods it can perform (display())

Objects

Java Object : Key Characteristics

✔️ Created using **new** keyword
✔️ Has its own **memory**
✔️ Can access class attributes and methods using dot operator
✔️ Multiple objects can be created from one class

Class vs Object

Class	Object
Blueprint	Instance
Logical entity	Physical entity
No memory	Occupies memory

Complete Example

```
class Car {
    // Attributes
    String brand;
    int speed;

    // Method
    void drive() {
        System.out.println("Car is driving at " + speed + " km/h");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Car car = new Car();   // object

        car.brand = "Toyota";
        car.speed = 80;

        car.drive();
    }
}
```

Output:

Car is driving at 80 km/h

Types of Access Modifiers in Java

Access modifiers in Java define the visibility of classes, variables, methods, and constructors.

They control who can access what in a program.

Java has **four** access modifiers:

1. public
2. private
3. protected
4. *default* (no keyword)

Types of Access Modifiers in Java

Modifier	Same Class	Same Package	Subclass (diff pkg)	Everywhere
public	✔	✔	✔	✔
protected	✔	✔	✔	✘
default	✔	✔	✘	✘
private	✔	✘	✘	✘

Types of Access Modifiers in Java

Why Access Modifiers are Important?

- Provide **security**
- Support **encapsulation**
- Control **data visibility**
- Improve **code maintainability**

PES UNIVERSITY
CELEBRATING 50 YEARS

MCQ

1. What is a class in Java?

A. A real-world entity
B. An instance of an object
C. A blueprint for creating objects
D. A method

2. An object is:

A. A blueprint of a class
B. A variable
C. An instance of a class
D. A package

MCQ

3. Which keyword is used to create an object in Java?

A. create

B. new

C. class

D. object

4. How many objects can be created from one class?

A. Only one

B. Two

C. Depends on memory

D. Any number

MCQ

5. Which access modifier allows access from anywhere?

A. private

B. protected

C. default

D. public

6. Which access modifier is used when no keyword is specified?

A. public

B. protected

C. private

D. default

PES UNIVERSITY
CELEBRATING 50 YEARS

MCQ SOLUTION

1.C
2.C
3.B
4.D
5.D
6.D

THANK YOU

Shridevi A Sawant
Department of Computer Science and Engineering
shridevias@pes.edu

Object Oriented Analysis and Design UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

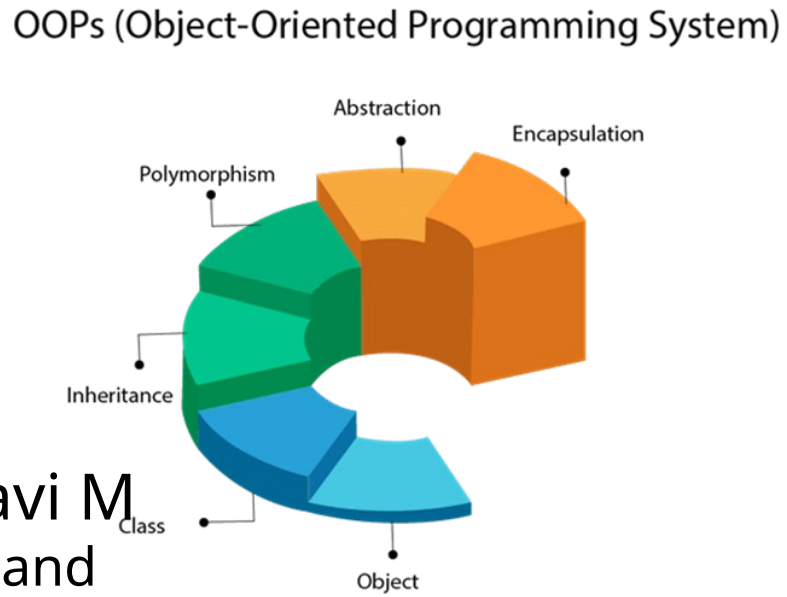
Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission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

UE23CS352B: Object Oriented Analysis and Design

Interfaces and Implementation

Prof . Shridevi A Sawant

Department of Computer Science and Engineering



Acknowledgements:

Prof. Mahitha G & Prof. Bhargavi M

Department of Computer Science and Engineering

Interface in Java

- An **interface** is a collection of abstract methods and constants that a class must implement.
- An interface in Java says what 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”.
- It is used to achieve **100% abstraction** and **multiple inheritance** in Java.

- An interface in Java specifies the method signatures and has no default implementation. So, these methods are **abstract** and also **public**

- **Note :** In Java 8+ version, interfaces can have methods with default implementation.

Interface : Syntax

Interface in Java is defined with the keyword '**interface**'

```
interface InterfaceName {
    // abstract methods
    // constants
}
```

Interface : Example

A bicycle's behaviour, 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 in Java

To implement the interface, the class 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.

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

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

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

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

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

Interface in Java

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

It is possible that there could be a number of implementations for the same interface.

Object Oriented Analysis and Design with Java

Interface vs Class

Feature	Class	Interface
Keyword used	class	interface
Object creation	✔️ Yes	❌ No
Variables	Instance variables allowed	Only constants (public static final)
Methods	Concrete + abstract	Abstract (default/static allowed from Java 8)
Method implementation	Can exist	Must be provided by implementing class
Inheritance	Extends **one** class	Class can implement **multiple** interfaces
Constructor	✔️ Allowed	❌ Not allowed
Access modifiers for methods	Any	Always public
Multiple inheritance	❌ Not supported	✔️ Supported
Purpose	Represents **what an object is**	Represents **what an object can do**

Interface in Java

- **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 be 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.

Interface in Java

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

◦ Definitely NO. (In Java 9+, yes)

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?** ◦

Yes, a class in Java can implement more than one interface using the implements keyword, separated by commas.

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

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

MCQ

1. **What does an interface in Java primarily specify?**
   A. How a class implements methods
   B. What a class must do
   C. Memory allocation rules
   D. Object creation logic

2. **In Java, methods declared in an interface are by default:**
   A. private and abstract
   B. protected and abstract
   C. public and abstract
   D. static and final

MCQ

3. Which concept does an interface mainly help achieve?

A. Encapsulation
B. Abstraction
C. Polymorphism
D. Compilation

4. What does it mean if a class implements an interface but does not override all methods?

A. The class is concrete
B. The class is final
C. The class remains abstract
D. The code will not compile

MCQ

5. Can an interface be instantiated directly in Java?

A. Yes
B. No
C. Yes, using new keyword
D. Only using anonymous classes

6. Can an interface contain data members?

A. No
B. Yes, mutable
C. Yes, static and final
D. Yes, instance variables

MCQ SOLUTION

1.B
2.C
3.B
4.C
5.B
6.C

THANK YOU

Prof. Shridevi A Sawant

Department of Computer Science and Engineering

shridevias@pes.edu

Object Oriented Analysis and Design (UE23CS352B)

Prof Shridevi A Sawant
Department of Computer Science and Engineering

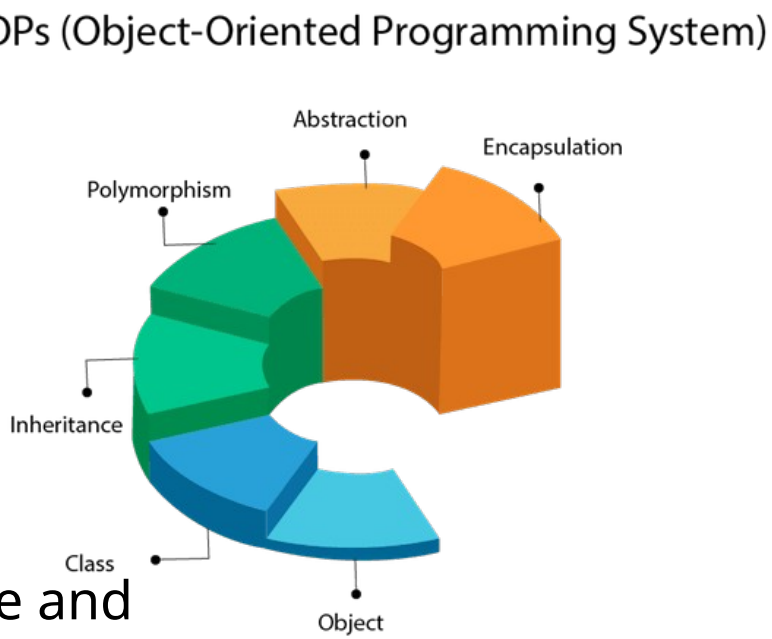
Object Oriented Analysis and Design using java

Constructors, Destructors and Garbage Collector

Prof. Shridevi A Sawant

Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G
Department of Computer Science and Engineering



Object Oriented Analysis and Design using Java

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**.
- 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: 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

Object Oriented Analysis and Design using Java

Constructor - Example

```
class Rect {
     int a; int b;

    Rect() {
        System.out.println("this is constructor");
    }

    void disp() {
        System.out.println("disp");
    }
}
```

```
public class Demo {

    public static void main(String[] args) {

        Rect r = new Rect();   // constructor is called here

        // r.rect();  //  cannot explicitly invoke constructor

        r.disp();

        System.out.println(r.a);
        System.out.println(r.b);
     }
}
```

```
Output -
this is constructor
disp
0
0
```

Object Oriented Analysis and Design using Java

Constructor - 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 Analysis and Design using Java

Parameterized Constructor - Example

- Constructor **accepts parameters that are u**sed to **initialize object data**
- Uses this to avoid name conflict

```
class Student {
    int id;
    String name;

    // Parameterized constructor
    Student(int id, String name) {
        this.id = id;
        this.name = name;
    }

    void display() {
        System.out.println(id + " " + name);
    }
}
```

Copy Constructor - Example

- Copy constructor takes **object of the same class** as argument
- Prevents unwanted **reference sharing**

```
class Student {
    int id;
    String name;

    // Copy constructor
    Student(Student s) {
        this.id = s.id;
        this.name = s.name;
    }
}
```

Object Oriented Analysis and Design using Java

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 Analysis and Design using Java

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 the class hierarchy.
- Called just prior to garbage collection and not called when an object goes out of scope
- Execution is not guaranteed
- **Deprecated since Java 9**

MCQ

1. **What happens if a class does not define any constructor?**
   A. Compilation error occurs
   B. Object cannot be created
   C. Java provides a default constructor
   D. Constructor returns null

2. **Which of the following correctly describes a parameterized constructor?**
   A. Constructor without arguments
   B. Constructor used only for copying objects
   C. Constructor that initializes object with given values
   D. Constructor that returns an object

MCQ

3. **What is the result if a constructor is declared private?**
   A. Object can be created anywhere
   B. Object can be created only inside the same class
   C. Compiler creates a default constructor
   D. Constructor becomes static

4. **Which of the following is NOT a valid characteristic of a constructor?**
   A. It initializes an object
   B. It has the same name as the class
   C. It can have parameters
   D. It has an explicit return type

MCQ

5. What is Java Garbage Collection?

A. Manual removal of objects from memory
B. Process of removing unused objects from memory automatically
C. Allocation of memory to objects
D. Compilation of Java programs

6. Which of the following objects are eligible for garbage collection?

A. Objects not referenced by any part of the program
B. Objects referenced by active variables
C. Objects that are currently in use
D. Objects created using new keyword

MCQ SOLUTION

1.C
2.C
3.B
4.D
5.B
6.A

PES UNIVERSITY
CELEBRATING 50 YEARS

Reference Book

R3: “Java the Complete Reference”, Herbert Schildt ,McGraw-Hill ,11th Edition, 2018.

THANK YOU

Shridevi A Sawant
Department of Computer Science and Engineering
shridevias@pes.edu

Object Oriented Analysis and Design - UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design
UE23CS352

Parameter Passing –
Value Types and Reference Types

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G & Bhargavi M
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Agenda

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

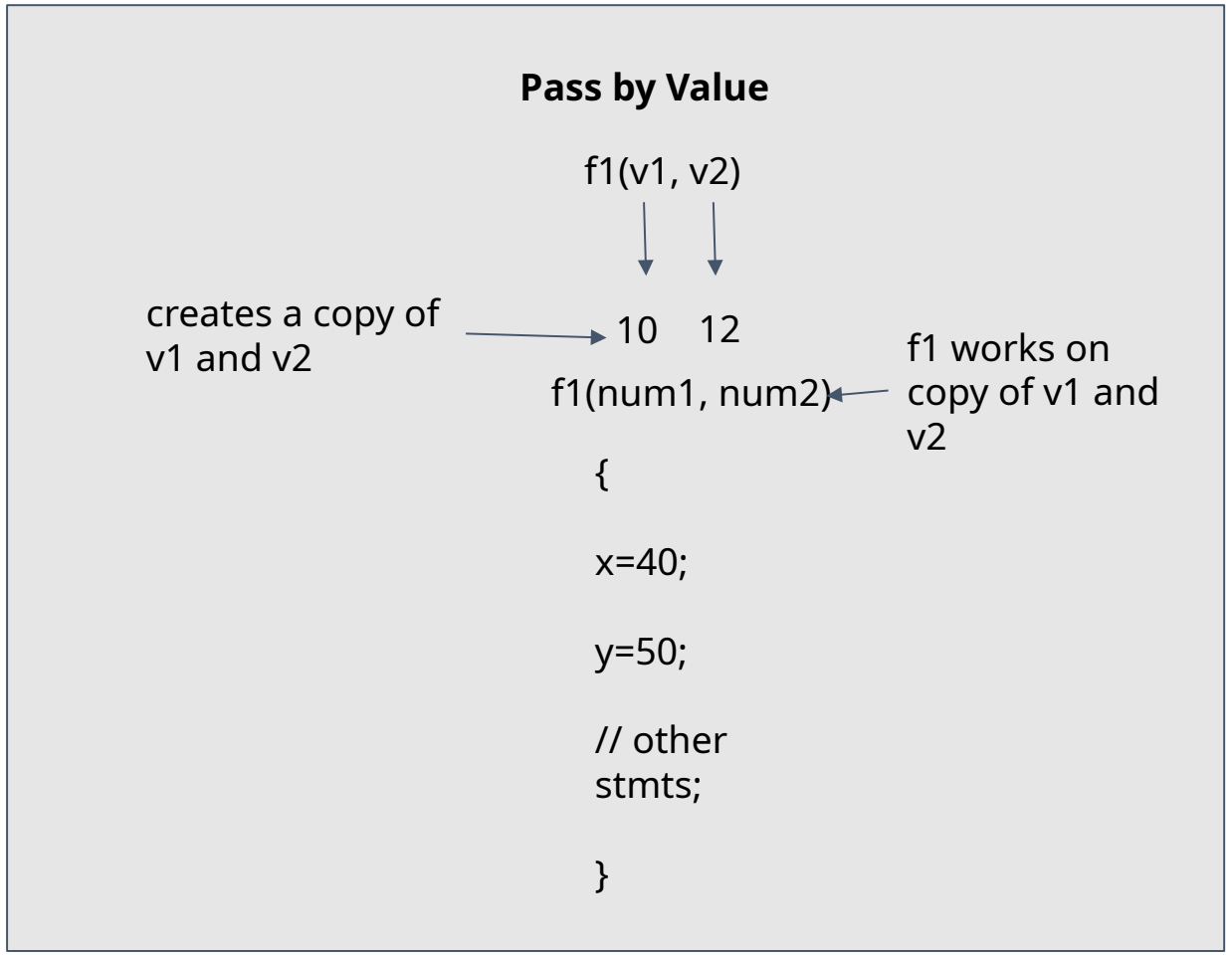
Parameter Passing

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.

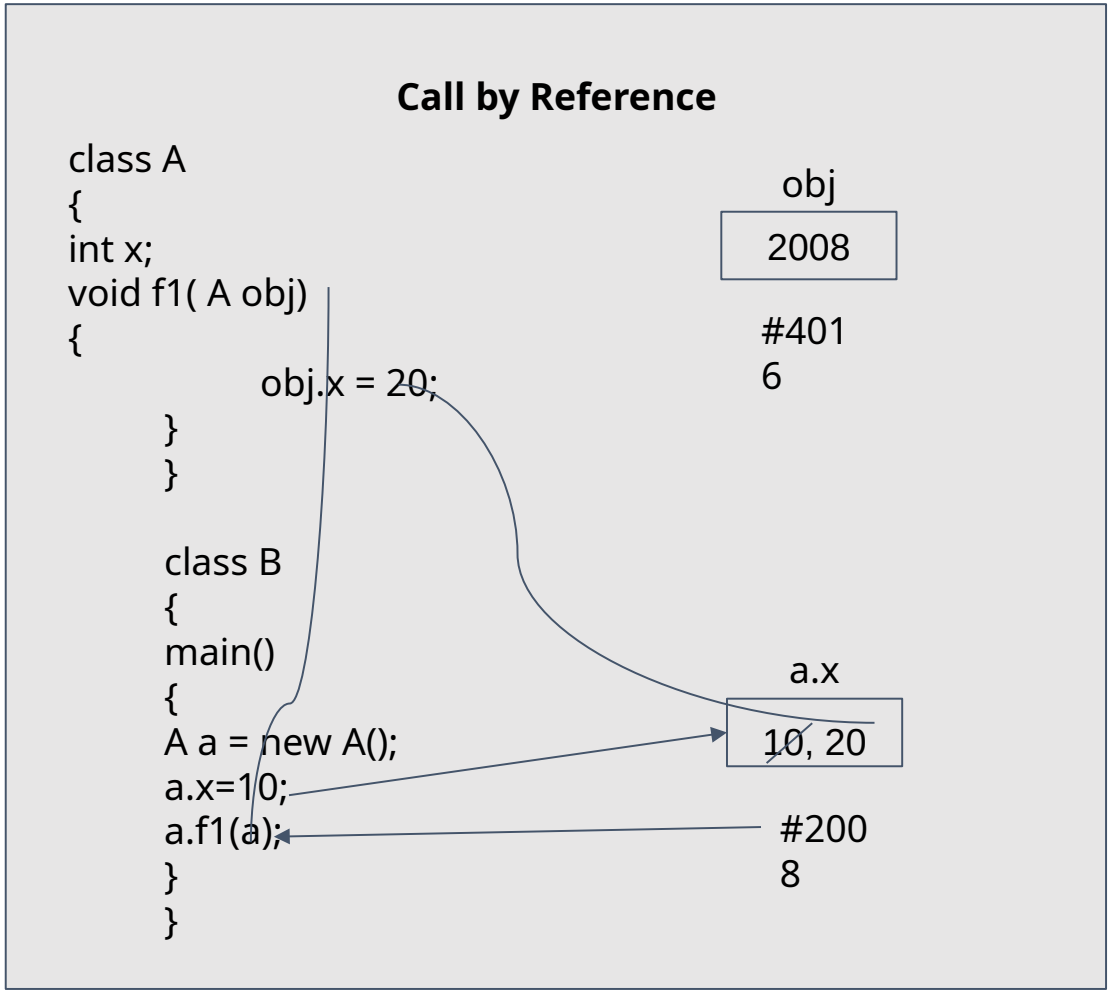


Object Oriented Analysis and Design

Parameter Passing

References- alias

- **Non-primitive type variables store references to objects**
- 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

Reference vs Object

Term	Meaning
Object	Actual data in heap memory
Reference	Memory address of the object
Variable	Stores the reference value

```
Student s = new Student();

(variable) s ──► (reference on stack) ──► Student object (on heap)
```

Object Oriented Analysis and Design

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;  }
}
```

Object Oriented Analysis and Design

1. **What happens to an argument when data is passed to a method in Java?**
   A.The argument is deleted
   B. The argument is copied to the parameter
   C. The argument becomes global
   D. The argument is ignored
2. **In pass by value, what happens when the formal parameter is modified?**
   A. Changes are reflected back to caller
   B. Caller variable is destroyed
   C. Changes affect only the formal parameter
   D. Reference is shared
3. **Which of the following data types in Java are passed as references?**
   A. Primitive types
   B. Value types
   C. Non-primitive types
   D. Constants
4. **When are changes to a reference type reflected back to the caller?**
   A. When reference is reassigned
   B. When object data is modified
   C. When method returns null
   D. When primitive data is used

MCQ SOLUTION

1.B
2.C
3.C
4.B

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Enginee

Object Oriented Analysis and Design - UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Overloading of Methods

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Priya Badarinath
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Agenda

1. Introduction
2. Coding examples

Object Oriented Analysis and Design
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

Object Oriented Analysis and Design

Overloaded methods

Overloaded methods:

- Have the same name
- Have different parameter lists
  - Number of parameters
  - Type of parameters
  - Order of parameters

✔️ Compiler decides which method to call (compile-time polymorphism) based on the method signature **: method name + parameter list (number, type, order)**

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.

Object Oriented Analysis and Design

Method Overloading : Example

```
class Calculator {
    // Method with two int parameters
    int add(int a, int b) {
        return a + b;
    }
    // Method with three int parameters
    int add(int a, int b, int c) {
        return a + b + c;
    }
    // Method with double parameters
    double add(double a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
       Calculator calc = new Calculator();

        System.out.println(calc.add(10, 20));
        System.out.println(calc.add(10, 20, 30));
        System.out.println(calc.add(5.5, 2.5));
    }
}
```

Output :

30
60
8.0

Object Oriented Analysis and Design

Method Overloading

MCQ

1. **What is method overloading in Java?**
   A. Defining methods with same name and same parameters
   B. Defining methods with different names
   C. Defining multiple methods with the same name but different argument lists
   D. Defining methods in different classes
2. **Method overloading is also known as:**
   A. Runtime polymorphism
   B. Late binding
   C. Dynamic polymorphism
   D. Compile-time polymorphism
3. **Which of the following best explains early binding?**
   A. Method call resolved during execution
   B. Method call resolved at runtime
   C. Method call resolved during compilation
   D. Method call resolved by JVM dynamically
4. **Which of the following is a key advantage of method overloading?**
   A. Reduces code reuse
   B. Increases complexity
   C. Allows same method name for different data types
   D. Prevents compilation

Object Oriented Analysis and Design

Method Overloading

MCQ SOLUTION

1.C
2.D
3.C
4.C

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Enginee

shridevias@pes.edu

Object Oriented Analysis and Design (UE23CS352)

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

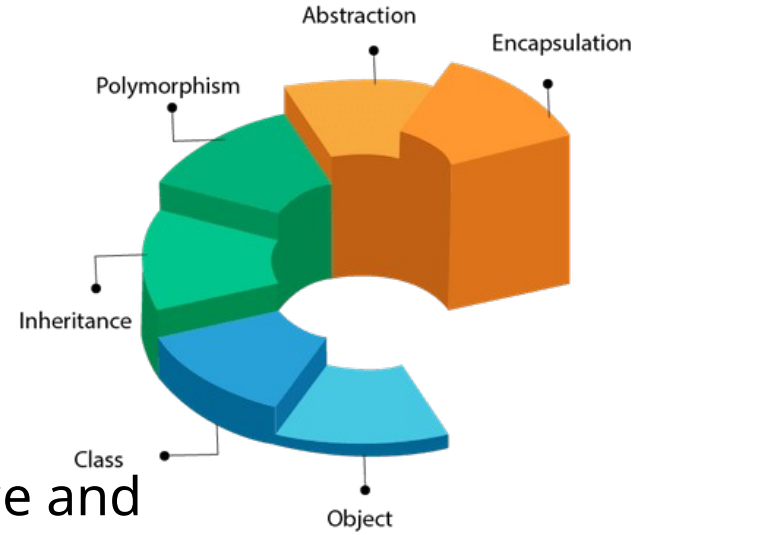
Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission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

Object Oriented Analysis and Design

Class Attributes and Methods

Prof. Shridevi A Sawant

Department of Computer Science and Engineering



Acknowledgements:
Prof. Mahitha G
Department of Computer Science and Engineering

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

Attribute Types

There are two types of attributes:

Instance Variables

Instance variables are attributes which require an object of its class to be created before it can be used. To access a instance variable, we have to create an object of the class in which the attribute is defined.

Static Variables

Static attributes are the variables that can be accessed without creating an object of class. They are referenced by the class name itself or reference to the object of that class.

By default, attributes are said to be instance attributes.

Object Oriented Analysis and Design

Instance attributes and methods : Example

```
class Example1 {

    public static void main(String[] args) {

        // create an instance of the class
        Sample obj1 = new Sample();

        // calling an instance method
        obj1.setName("Ramu");
        System.out.println(obj1.name);
    }
}

class Sample {

    String name = ""; // instance attribute

    public void setName(String s) {
        name = s;
    }
}
```

Object Oriented Analysis and Design

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

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

Object Oriented Analysis and Design
Static Method and Attribute: Example

```
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 using object
         Sample obj = new Sample();
         obj.setName("manish");
         System.out.println(obj.name);
    }
}

class Sample {
    public static String name = "";

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

Object Oriented Analysis and Design
Static Attribute vs Instance Attribute

Feature	Static Attribute	Instance Attribute
Belongs to	Class	Object
Copies in memory	One	One per object
Accessed using	Class name	Object
Value shared	Yes	No
this keyword	❌ Not allowed	✅ Allowed

Object Oriented Analysis and Design

Static vs Instance Method

Feature	Static Method	Instance Method
Belongs to	Class	Object
Called using	Class name	Object
Object required	❌ No	✅ Yes
Access instance variables	❌ No	✅ Yes
Access static variables	✅ Yes	✅ Yes
Use of this	❌ Not allowed	✅ Allowed
Memory	One copy	One per object

Object Oriented Analysis and Design

Static Block

Java static block

Is used to initialize the static data member.

It is executed before the main method at the time of class loading and is executed exactly once.

Static block gets executed exactly once, when the class is first loaded

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 Class

A class can be made static only if it is a **nested class**. We cannot declare a top-level class with a static modifier but can declare nested classes as static. Such types of classes are called Nested static classes. Nested static class doesn’t need a reference of Outer class. In this case, a static class cannot access non-static members of the Outer class.

MCQ

1. **Which of the following correctly describes an instance method?**
   A. Can be called without creating an object
   B. Must be called using the class name
   C. Requires an object of the class to be created
   D. Is executed only once
2. **Which of the following is true about static methods?**
   A. They require object creation
   B. They can be called only inside main
   C. They can be called using the class name
   D. They are inherited automatically
3. **By default, methods in Java are considered as:**
   A. Instance methods
   B. Abstract methods
   C. Static methods
   D. Final methods

4. **Which of the following can use the static keyword?**
   A. Variables only
   B. Methods only
   C. Classes only
   D. Variables, methods, blocks, and nested classes
5. **When is a static block executed?**
   A. After main method execution
   B. Before object creation
   C. Before the main method during class loading
   D. When garbage collection occurs
6. **Which statement is true about class variables (static variables)?**
   A. Each object has its own copy
   B. They are shared by all instances of the class
   C. They are stored inside objects
   D. They must be initialized inside constructors

MCQ SOLUTION

1.C
2.C
3.A
4.D
5.C
6.B

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Engineering
shridevias@pes.edu

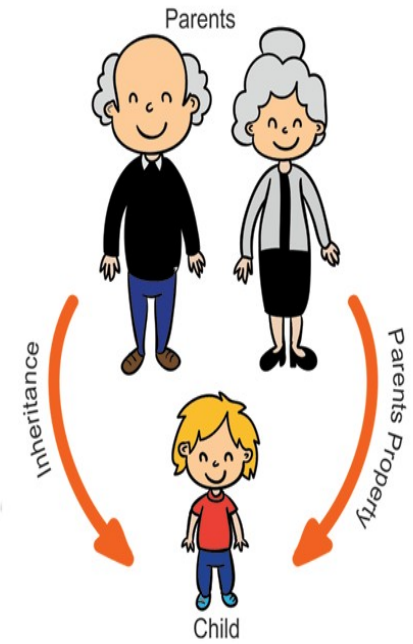
Object Oriented Analysis and Design UE23CS352B

Prof. Shridevi A Sawant
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

Object Oriented Analysis and Design

Inheritance

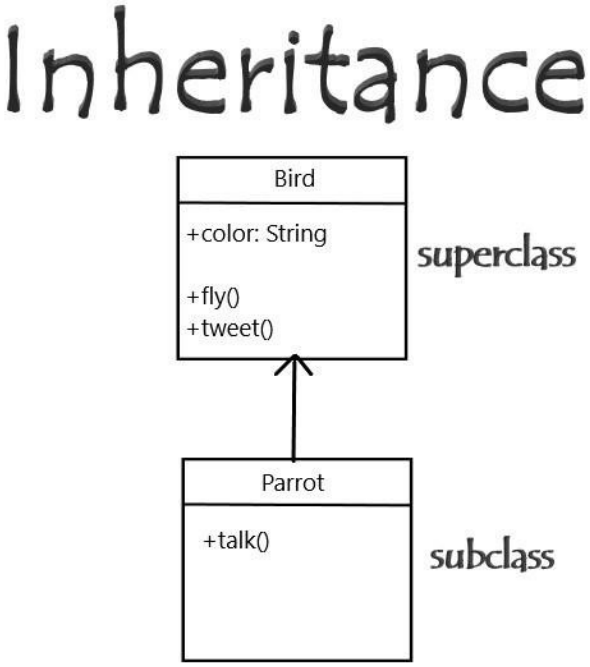


Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G
Department of Computer Science and Engineering

Inheritance

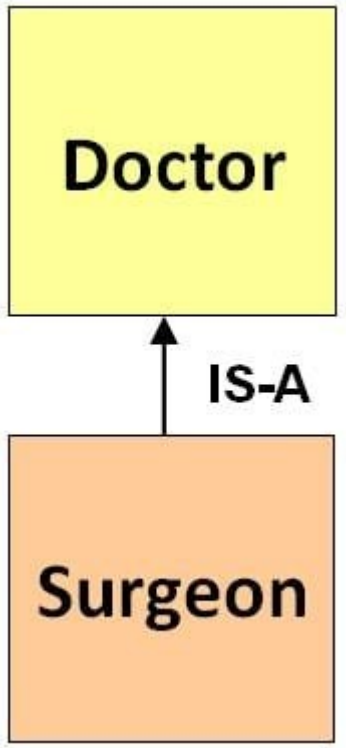
- Inheritance is an important pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class.
- Inheritance represents **IS-A** relationship which is also known as a parent-child relationship
- Acquiring 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

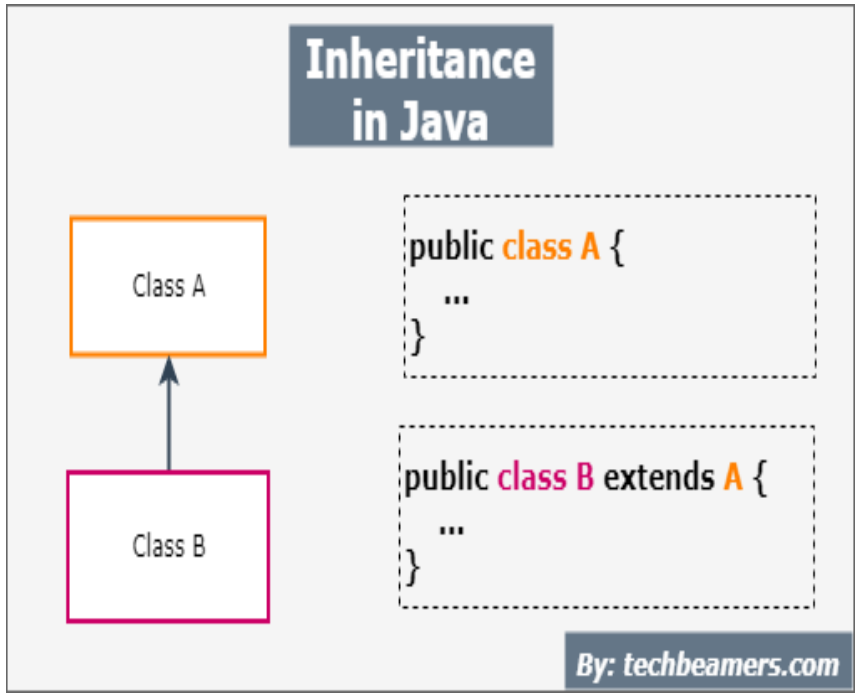


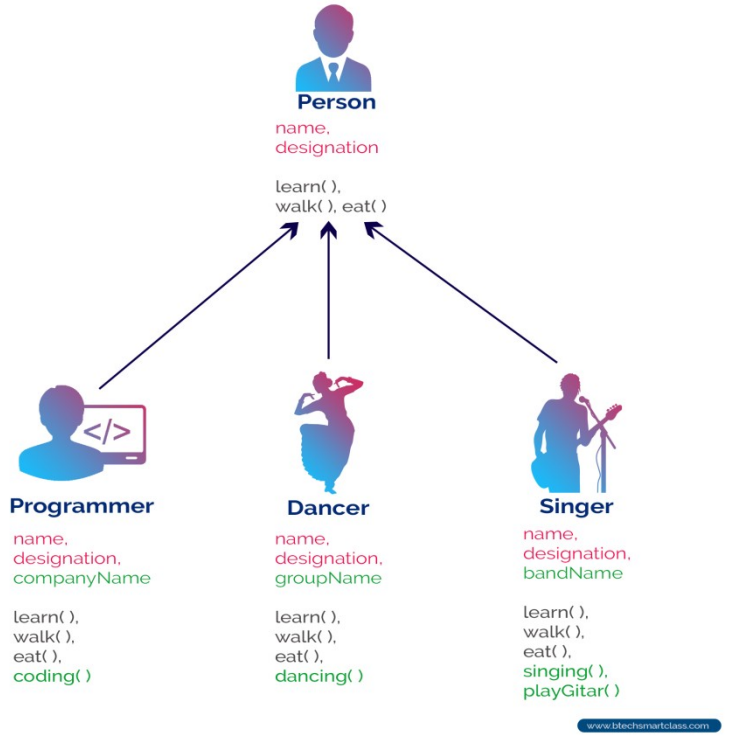
Object Oriented Analysis and Design

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

Inheritance in Java

```
class Teacher {
   String designation = "Teacher";
   String collegeName = "PESU";
   void teach(){
            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);
```

OUTPUT

```
PESU
Teacher
Java
Teaching
```

Cannot inherit from more than one Super Class

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.

Types of Inheritance

- Single Level Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance (Not in Java)
- Hybrid Inheritance (Partially in Java)

Types of Inheritance -Example

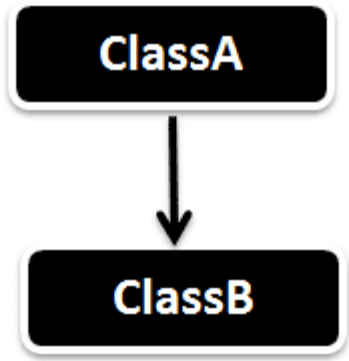
Single Level Inheritance:

```
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);
    }
}
```

One child class inherits from one parent class.

OUTPUT

```
Programmer salary is: 40000.0
Bonus of Programmer is: 10000
```



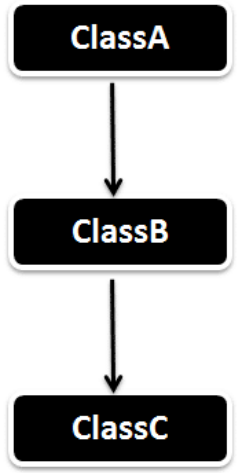
Types of Inheritance

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();
}
}
```

OUTPUT

```
weeping...
barking...
eating...
```



instanceof is a keyword used for checking if a reference variable contains a given type of object reference or not. It returns boolean value.

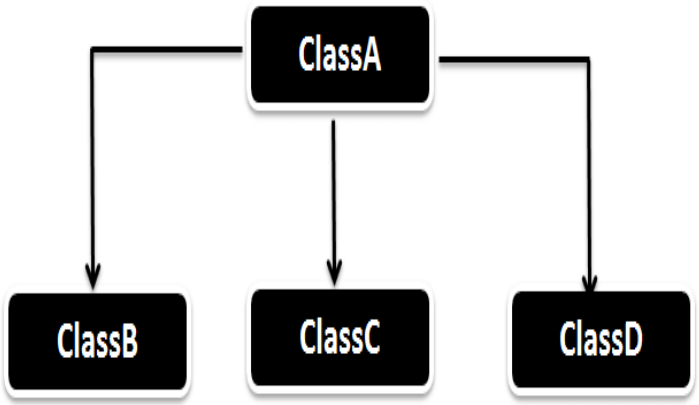
Types of Inheritance

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
}
```

OUTPUT

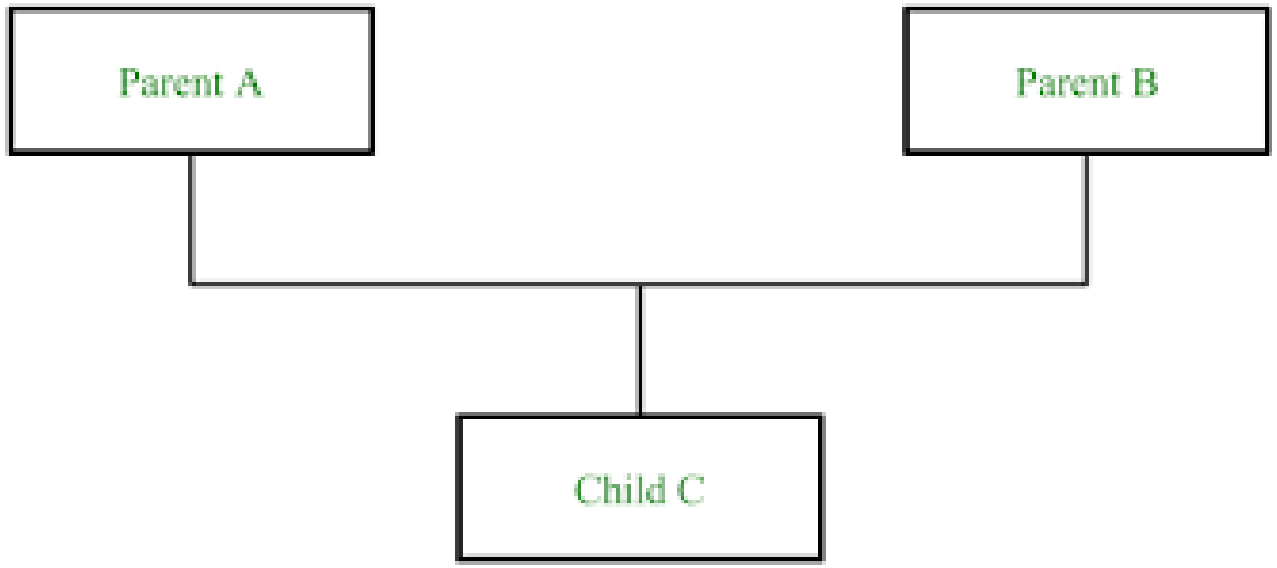
```
meowing...
eating...
```



Types of Inheritance

Multiple Inheritance

- One class inherits from more than one parent class.
- **Java does NOT support multiple inheritance using classes** (to avoid ambiguity problem).
- But Java achieves it using **interfaces.**



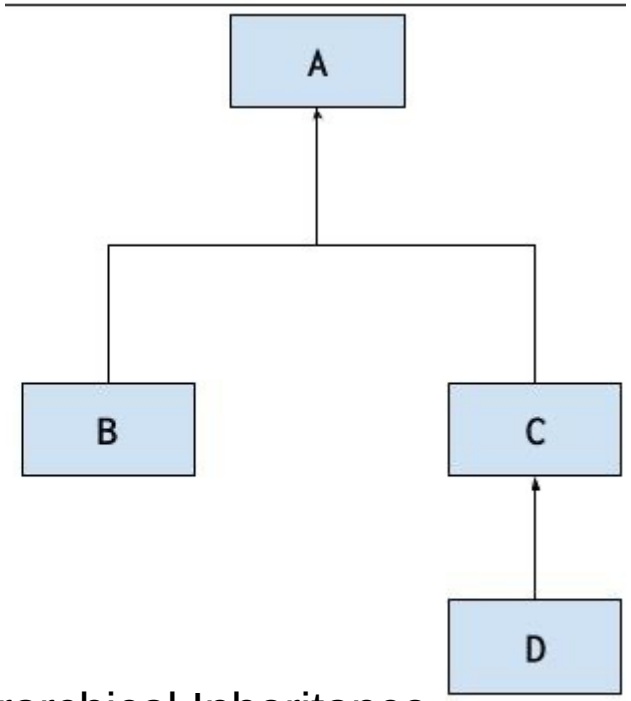
Multiple Inheritance via Interface

```
interface A {
    void show();
}

interface B {
    void display();
}

class C implements A, B {
    public void show() {}
    public void display() {}
}
```

Types of Inheritance

Hybrid Inheritance

- Two or more types of inheritance—such as single, multiple, multilevel, or hierarchical—are combined to create a complex class hierarchy
- It typically merges patterns like **hierarchical and multiple** inheritance or **single and multilevel** inheritance.
- Java does not support multiple inheritance via classes; must use interfaces to achieve hybrid structures.



A, B, C - > Hierarchical Inheritance

A, C, D → Multilevel Inheritance

Object Oriented Analysis and Design

Constructor and Inheritance

When the object of subclass is created, constructor of sub class by default invokes the **default constructor of super class** . Hence, in inheritance the objects are constructed top-down.
The superclass constructor can be called explicitly using the **super** keyword, but it should be first statement in a constructor.
The **super** keyword refers to the superclass, immediately above of the calling class in the hierarchy.

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

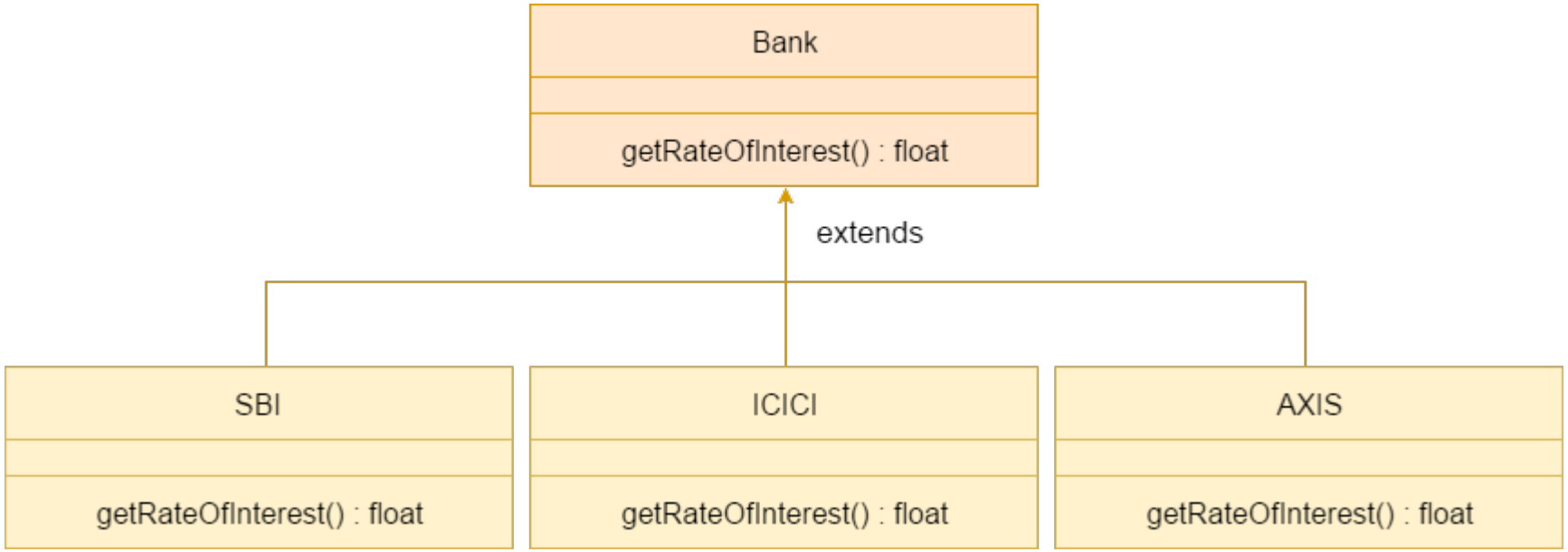
1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Object Oriented Analysis and Design

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.



Method Over-riding - Example program

An Example of Java Method Overriding

Hands-On

MCQ

1. **What does inheritance represent in Java?**
   A. HAS-A relationship
   B. USES-A relationship
   C. IS-A relationship
   D. PART-OF relationship
2. **The class whose properties are inherited is called:**
   A. Subclass
   B. Superclass
   C. Derived class
   D. Child class
3. **Which keyword is used in Java to achieve inheritance?**
   A. implements
   B. inherits
   C. extends
   D. instanceof

4. **What is method overriding in Java?**
   A. Defining a method with same name but different parameters
   B. Redefining a superclass method in the subclass
   C. Calling superclass methods
   D. Hiding method

5. **Method overriding supports which type of polymorphism?**
   A. Compile-time polymorphism
   B. Static polymorphism
   C. Early binding
   D. Runtime polymorphism

MCQ SOLUTION

1.C
2.B
3.C
4.B
5.D

THANK YOU

Prof. Shridevi A Sawant

Department of Computer Science and Engineering

shridevias@pes.edu

Object Oriented Analysis and Design - UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Abstract class

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Abstract class and Object class - Agenda

1. Introduction to Abstract class
2. Creation and Usage in Java
3. Coding examples – Demo

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

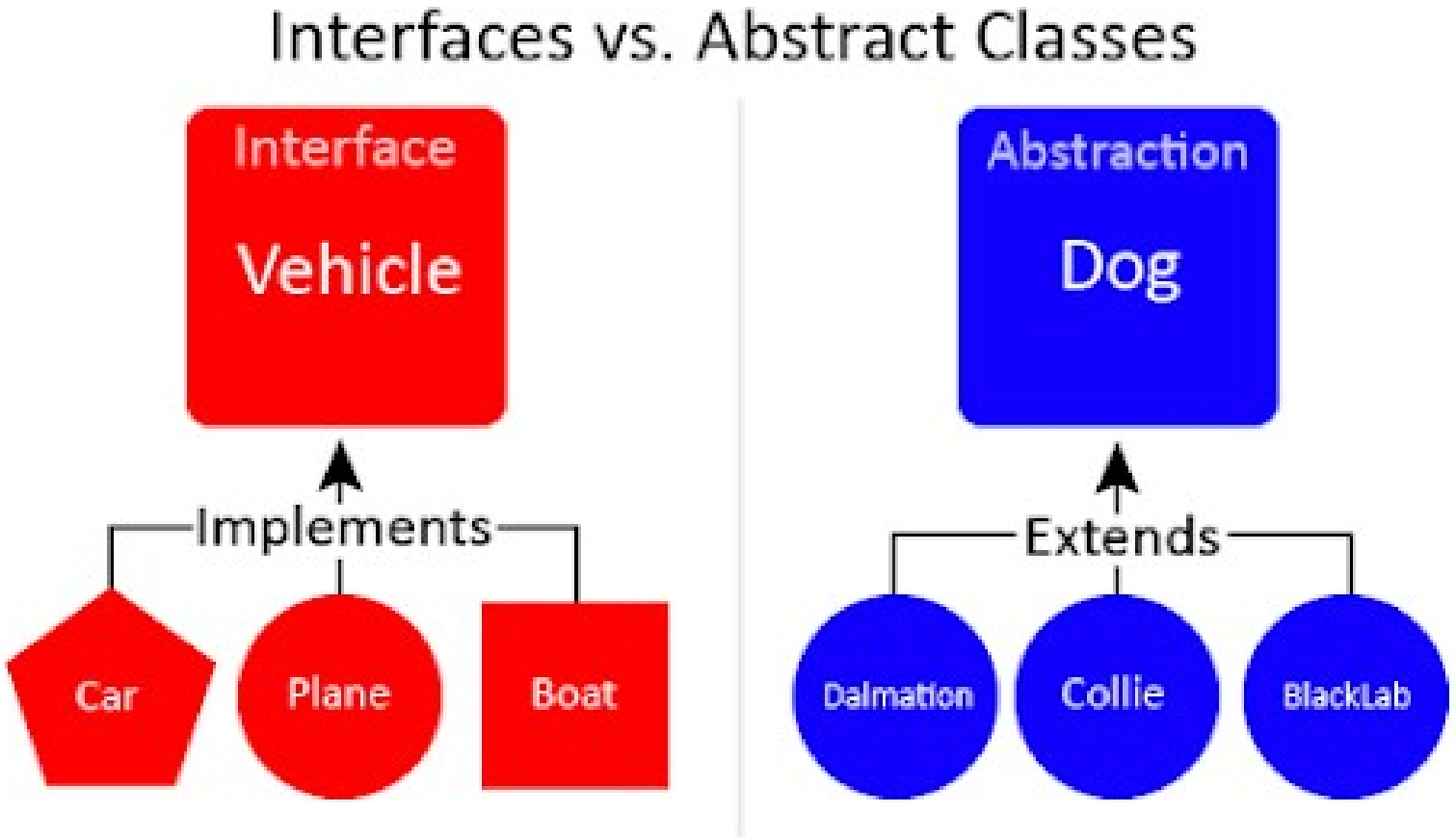
Object Oriented Analysis and Design

Abstract class: Example

```
abstract class Figure {

    double dim1;
    double dim2;

    Figure(double a, double b) {
        dim1 = a;
        dim2 = b;
    }

    abstract double findArea();
}
```

```
class Rectangle extends Figure {
        Rectangle(double height, double width){
         super(height, width);
         }
          double findArea(){
             return dim1 * dim2
          }
}

class Triangle  extends Figure {
        Triangle(double base, double height){
         super(base, height);
         }
          double findArea(){
             return 0.5 * dim1 * dim2
          }
```

Abstraction Vs Interface



Object Oriented Analysis and Design

Abstraction Vs Interface

Feature	Abstract Class	Interface
Keyword	abstract class	interface
Object Creation	Cannot create object	Cannot create object
Methods	Can have abstract + concrete methods	By default abstract methods (can have default & static methods from Java 8)
Variables	Can have instance variables	Only public static final (constants)
Constructors	✅ Allowed	❌ Not allowed
Access Modifiers	Can use any (private, protected, public)	Methods are public by default
Multiple Inheritance	❌ Not supported (for classes)	✅ A class can implement multiple interfaces
Purpose	Partial abstraction	Full abstraction (possible)

MCQ

1. **What is the primary purpose of an abstract class?**
   A. To support multiple inheritance
   B. To provide implementation reuse
   C. To define only method signatures
   D. To improve execution speed
2. **Why is an abstract class used as a superclass?**
   A. To avoid inheritance
   B. To define a generalized form shared by subclasses
   C. To enforce object creation
   D. To remove abstraction
3. **What is true about abstract methods?**
   A. They have a method body
   B. They are private
   C. They have no method body
   D. They must be static

MCQ

4. Can an abstract class be instantiated?

A. Yes
B. No
C. Only using inheritance
D. Only using constructors

5. What must a subclass do when it extends an abstract class?

A. Nothing
B. Implement all abstract methods or be abstract
C. Override only static methods
D. Implement interfaces

6. Why can an abstract class not be declared final?

A. Because it has abstract methods
B. Because it cannot be instantiated
C. Because it must be extended
D. Because it is slow

MCQ SOLUTION

1.B
2.B
3.C
4.B
5.B
6.C

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Enginee

Object Oriented Analysis and Design UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Single Rooted Hierarchy and Object class

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgement: Prof. Mahitha G
Department of Computer Science and Engineering

Single Rooted Hierarchy

- Same Base Class
- Common Interface
- It enables easy memory management
- Simplifies argument passing, allowing generic handling of different object types.

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

Single Rooted Hierarchy

Implementation of Garbage Collector is became easy since required implementation is provided in the base class, enabling to send messages to every object.

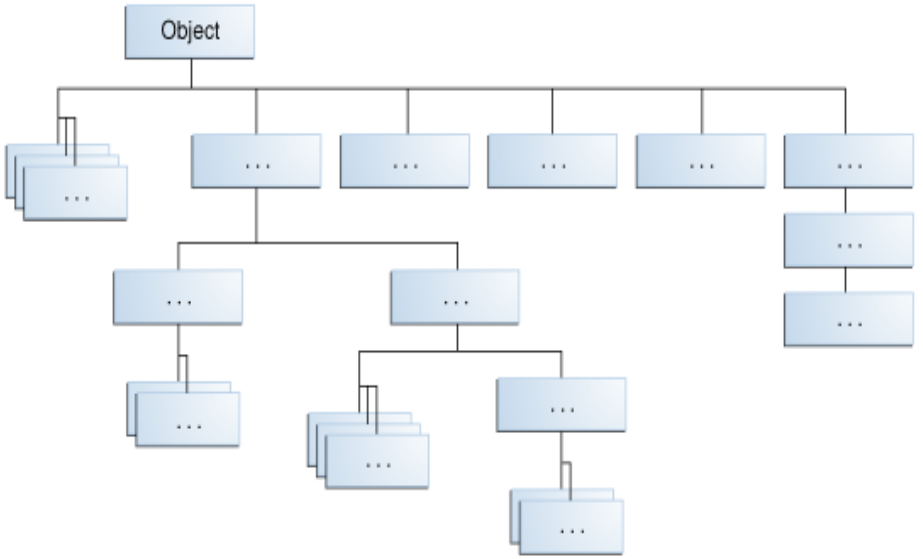
All objects in Java are inherited from same base class called 'Object'.

In Java all objects have common interface to implement and it makes implementation of Garbage collector lot easier in Java.

Object Oriented Analysis and Design using Java

Object class -Introduction

- **Object** class defined by Java is a super class of all other classes, in the absence of any other explicit superclass
- Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class
- A reference variable of type Object can refer to an object of any class
- This is defined in the **java.lang** package



Object Oriented Analysis and Design using Java

Object class: Methods

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.

Object Oriented Analysis and Design using Java

Object class Methods

- **`public final Class<?> getClass()`**
- Returns the runtime class of this `Object`.
- **`public int hashCode()`**

Returns a hash code value for the object. This method is supported for the benefit of hash tables such as those provided by `HashMap`.

The general contract of `hashCode` is:

- Whenever it is invoked on the same object more than once during an execution of a Java application, the `hashCode` method must consistently return the same integer. This integer need not remain consistent from one execution of an application to another execution of the same application.

Object class Methods

public boolean equals(Object obj)

Indicates whether some other object is "equal to" this one.

The equals method implements an equivalence relation on non-null object references:

Purpose:

Default Behavior: Compares memory reference (same as ==).

Often overridden for content comparison.

Object Oriented Analysis and Design using Java

Object class

```
class Box {
    int width, height, depth;

    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;   // important
        return this.width == b2.width
                && this.height == b2.height
                && this.depth == b2.depth;}}
public class P2Object {
    public static void main(String[] args) {
        Box obj1 = new Box(3, 2, 1);
        Box obj2 = new Box(3, 2, 1);
        System.out.println(obj1 == obj2);
        System.out.println(obj1.equals(obj2)); }}
```

== Operator	equals() Method
Compares memory address (reference)	Compares content (logical equality)

Output

false
true

Object Oriented Analysis and Design using Java

Object class- Methods

public String toString()

Returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method.

The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object

Default return : *ClassName@hashcode*

Object Oriented Analysis and Design using Java

Object class

```
class Box {
    int width;
    int height;
    int depth;
    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 BoxDemo {
    public static void main(String[] args) {
         Box newObj = new Box(3, 2, 1);
        System.out.println(newObj); }
}
```

Output:

3 2 1

MCQ

1. **What is meant by *single-rooted hierarchy* in Java?**
   A. Every class can inherit from multiple base classes
   B. All classes inherit from a single base class
   C. Interfaces act as the root of inheritance
   D. Only abstract classes can be inherited
2. **Which class acts as the root of the Java class hierarchy?**
   A. Base
   B. Root
   C. Object
   D. Class

MCQ

3. **A reference variable of type Object can refer to:**
   A. Only Object instances
   B. Only primitive wrapper classes
   C. Any class object
   D. Only abstract class objects

4. **Which of the following methods returns the runtime class of an object?**
   A. getRuntime()
   B. getType()
   C. instanceof
   D. getClass()

MCQ

5. What should x.equals(null) return for any non-null reference value x?

A. true
B. false
C. NullPointerException
D. Compiler error

6. What is the default output format of Object.toString()?

A. Object memory address
B. Class name only
C. Class name + @ + hexadecimal hash code
D. Hash code only

MCQ SOLUTION

1.B
2.C
3.C
4.D
5.B
6.C

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Enginee

shridevias@pes.edu

Object Oriented Analysis and Design - UE23CS352B

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design

Array, List and Stack

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G & Bhargavi M
Department of Computer Science and Engineering

Object Oriented Analysis and Design
Collections - Arrays, List, Stack

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

Object Oriented Analysis and Design
Collections - Introduction

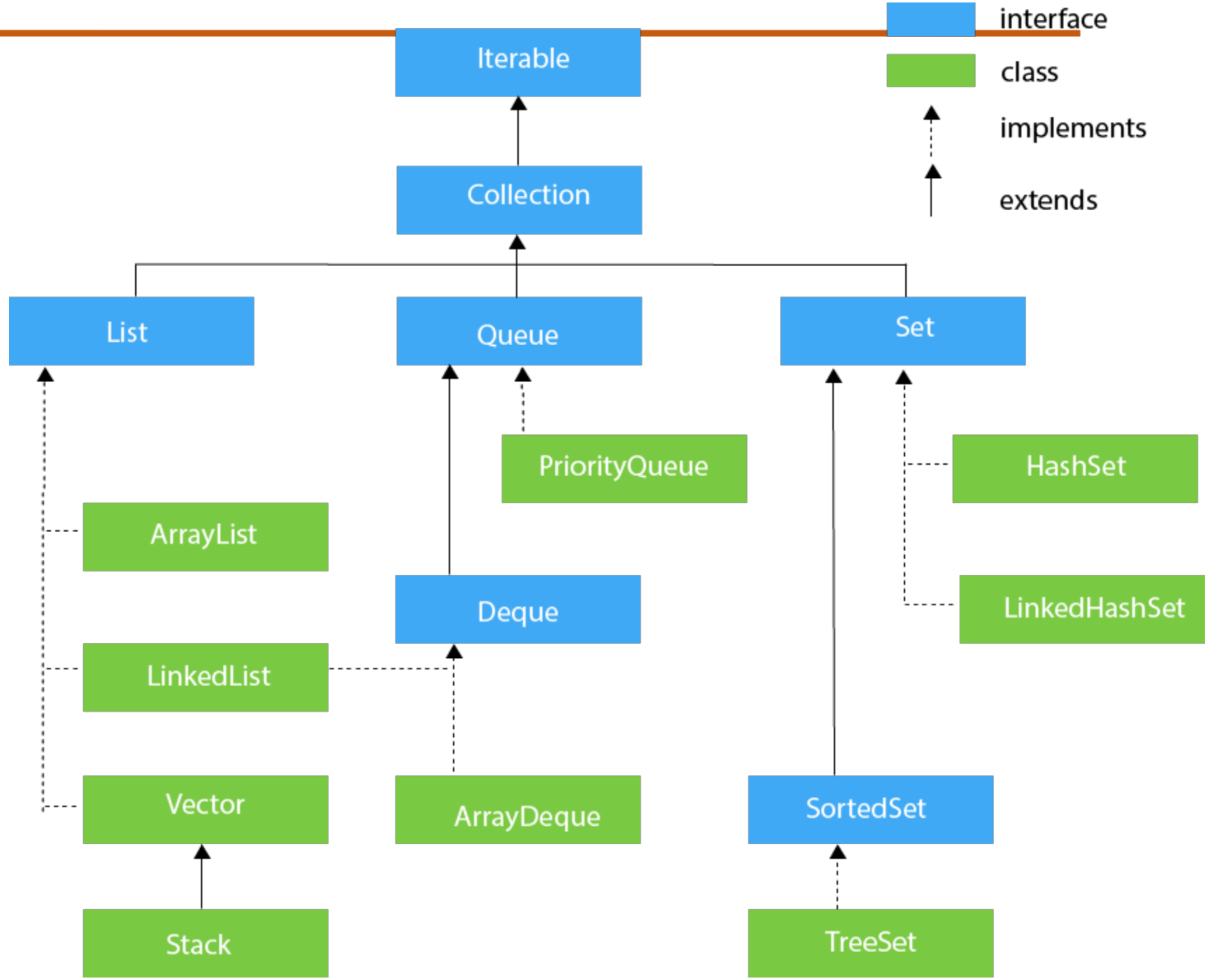
Collections framework. A *collection* is an object that represents a group of objects.

The primary advantages of a collections framework are that it:

- **Reduces programming effort** by providing data structures and algorithms so you don't have to write them yourself.
- **Increases performance** by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
- **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
- **Reduces the effort required to learn APIs** by requiring you to learn multiple ad hoc collection APIs.
- **Fosters software reuse** by providing a standard interface for collections and algorithms with which to

Object Oriented Analysis and Design

Collections - Introduction



Object Oriented Analysis and Design

Array - 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

Object Oriented Analysis and Design

Array Representation

Representation of 1 Dimensional Array

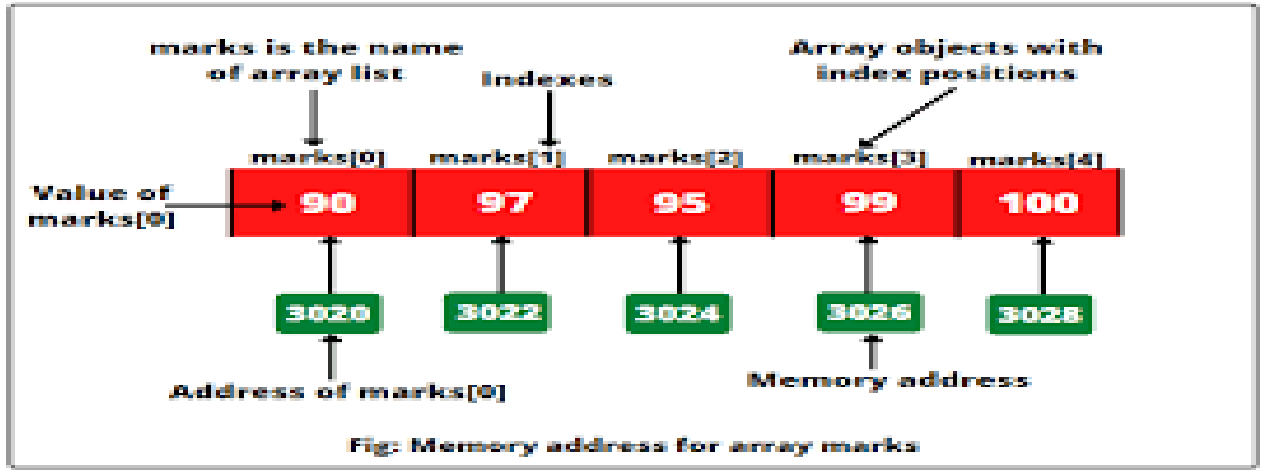


Fig: Memory address for array marks

Representation of 2 Dimensional Array

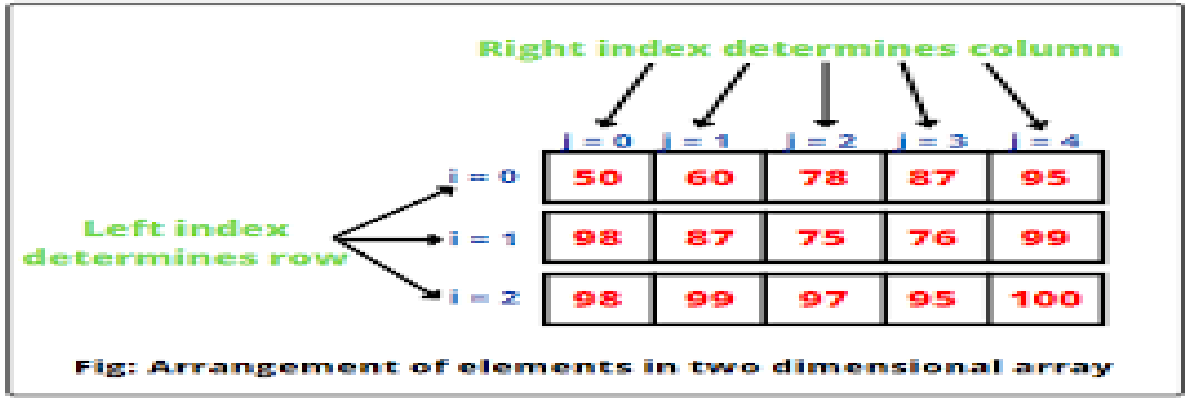


Fig: Arrangement of elements in two dimensional array

Object Oriented Analysis and Design

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){ ...}.

Object Oriented Analysis and Design

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

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 :

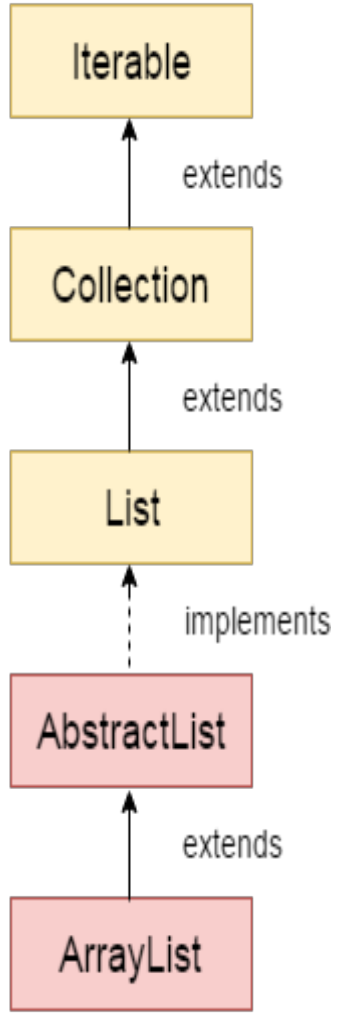
- `size()`, `clear()`, `add()`, `Add()`,`addAll()`, `contains()`, `containsAll()`, `equals()`, `hashCode()`, `isEmpty()`, `indexOf()`. etc...

Object Oriented Analysis and Design

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



Object Oriented Analysis and Design
Introduction to ArrayList Class in Java

Declaration :

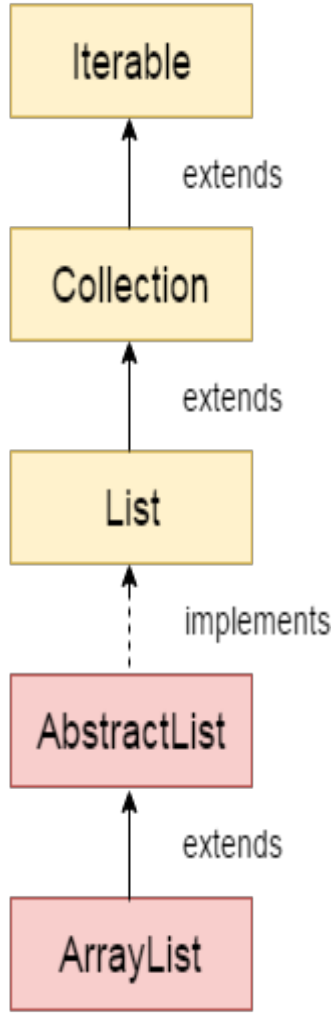
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable

Example for creating generic ArrayList:

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

Few Methods()

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



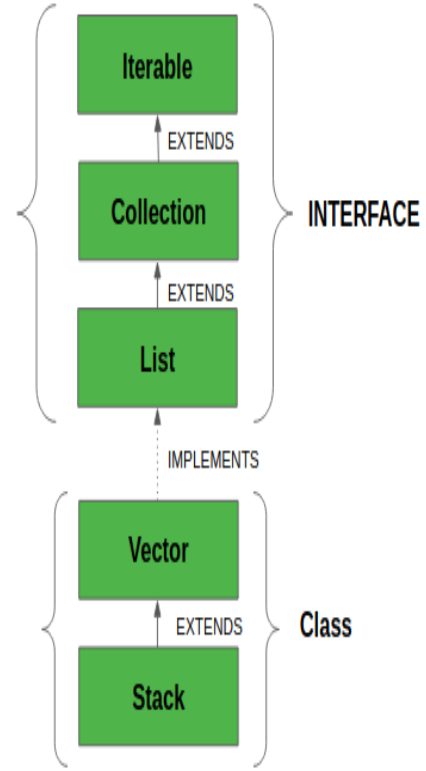
Object Oriented Analysis and Design

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.

Object Oriented Analysis and Design

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)

DEMO

MCQ

1. **What is a collection in Java?**
   A. A primitive data type
   B. A class for storing numbers
   C. An object that represents a group of objects
   D. A memory allocation technique
2. **How does the Collections Framework improve performance?**
   A. By using contiguous memory
   B. By compiler optimizations
   C. By providing high-performance implementations
   D. By removing algorithms

MCQ

3. **Which is a valid declaration of a one-dimensional array?**
   A. dataType arr{}
   B. dataType[] arr
   C. arr[dataType]
   D. array dataType

4. **Which class provides utility methods for arrays and belongs to java.util?**
   A. ArrayList
   B. Collections
   C. Arrays
   D. Objects

MCQ

5. **What feature does the List interface provide?**
   A. Unordered collection
   B. Index-based access
   C. Fixed size
   D. No duplicates
6. **Which classes implement the List interface?**
   A. HashSet and TreeSet
   B. ArrayList, LinkedList, Stack, Vector
   C. Queue and Deque
   D. Map and HashMap

MCQ SOLUTION

1.C
2.C
3.B
4.C
5.B
6.B

THANK YOU

Prof Shridevi A Sawant
Department of Computer Science and Enginee

shridevias@pes.edu

Object Oriented Analysis and Design

UE23CS352B

Prof. Shridevi A Sawant

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

UE23CS352B: Object Oriented Analysis and Design

OO Development Process, System Design and Frameworks

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G & Bhargavi M
Department of Computer Science and Engineering

Agenda

- OO Development Process
- OO Methodology
- Stages of OO methodology
- System Design
- Decisions – System Architecture
- Estimating performance
- Reuse Plan
- Libraries
- Patterns
- Frameworks

Object Oriented Development Process

- The essence is the **identification and organization of application concepts rather than their final representation in programming language.**
- Encourages software developers **to work and think in terms of application throughout the software life cycle**.
- Conceptual process independent of the programming language until final stage
- A way of thinking and it's greatest benefit comes from helping developers, customers express abstract concepts clearly and communicate them to each other.

Object Oriented Methodology

- Emphasis how to go about developing a system or application
- **Encourages and facilitates re-use of software components**.
- It employs **international standard Unified Modeling Language (UML) from the Object Management Group (OMG).**
- A system can be developed on a component basis, which enables the effective re-use of existing components, it facilitates the sharing of its other system components
- Asks the analyst to determine what the objects of the system are?, What responsibilities and relationships an object has to do with the other objects? and how they behave over time?

Stages of Object Oriented Methodology

- **System conception :** Software development begins with business analysis or users conceiving an application and formulating tentative Requirements.
- **Analysis** : The analyst must work with the requestor to understand the problem, because problem statements are rarely complete or correct.

  The analysis model is a precise abstraction of **what the desired system must do**, **not how it will be done.** It should not contain implementation decisions.

  Analyst must figure out the big picture of the entire application – for whom the application is to be developed, what problems the application will solve, when, where and why it will be needed.

  Analyst must also try and figure out the workflow of the application.

Stages of Object Oriented Methodology contd..

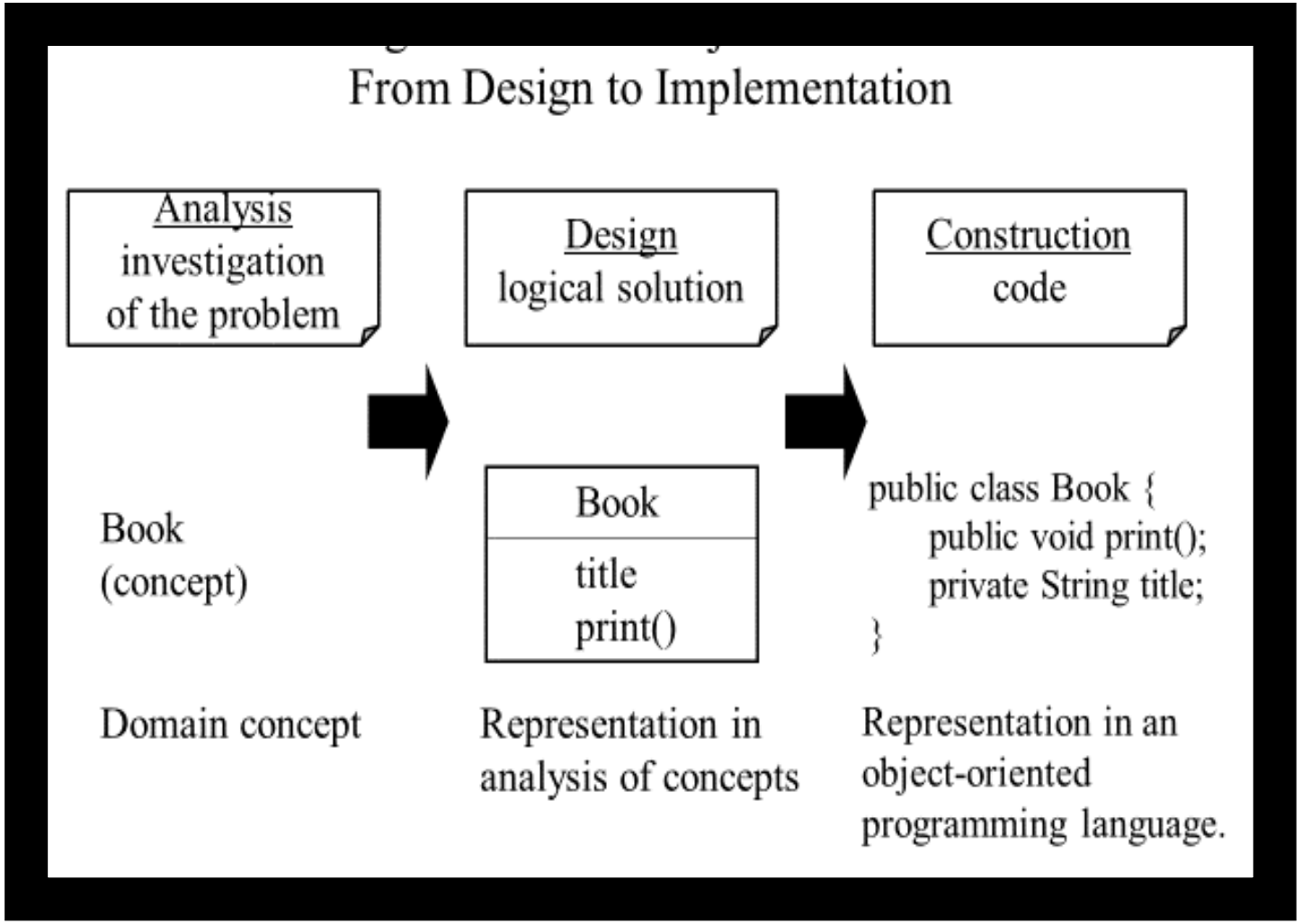
- **System design**: The development teams devise a **high – level strategy called the system**

architecture for solving the application problem.

- **Class design** : The class designer adds details to the analysis model in accordance with the system design strategy. The focus of class design is the **data structures and algorithms** needed to implement each class.
- **Implementation** : Implementers **translate the classes and relationships** developed during class design into **particular programming language, database or hardware**. During implementation, it is important to follow **good software engineering practice** so that traceability to the design is apparent and so that the

Object Oriented Analysis and Design

Sampl
e



System Design

- The first design stage for devising the basic approach to solve the problem.
- Developers make decisions about how the problem will be solved, first at a high level and then with more details – Overall **structure and style.**
- Need to apply **high level strategy – System Architecture** – for solving the problem and building a solution. It determines the **organization of the system into subsystems**. Also provides **the context for the detailed decisions** that are made in later stages.

Object Oriented Analysis and Design

Decisions - System Design

1. Estimating performance
2. Making a Reuse plan
3. Breaking system into sub-systems
4. Identifying Concurrency
5. Allocation of Sub Systems to hardware
6. Manage data storage
7. Handling global resources
8. Choosing a software control strategy
9. Handling boundary conditions
10. Set trade-off priorities
11. Select an architectural Style

System Design - Estimating performance

- Preparing a rough performance estimate – “**back of the envelope”** calculation
- Purpose is to **determine if the system is feasible** rather than achieving high accuracy
- Involves simplifying assumptions (i.e., assume factors).
- Don’t worry about details – just approximate, estimate and guess
- Example: ATM Network
  - Bank has 40 branches and no. of terminals.
  - On a busy day, half the terminals are busy at once.
  - Suppose each customer takes one min to perform a session and most transactions involve a single transaction. i.e., 40 transactions a minute.
  - You can perform similar estimates for data storage.

System Design – Reuse Plan

- **Reuse –** Advantage of OO but it does not happen automatically.
- Two different aspects of reuse - **Using existing things and Creating reusable things.**
- Much easier to reuse existing things than to design new things.
- Most developers reuse existing things and a small fraction of developers create new things.
- Reusable things include **Models, Libraries, Frameworks and Patterns.**

Reuse Plan - Libraries

- Library is a collection of classes that are useful in many contexts.
- Classes must be carefully organized, so that users can find them.
- Classes must have accurate and thorough description to help users determine their relevance.
- Several qualities of good class libraries:

Coherence: Organized about a few well focused themes;

Completeness: Provide complete behavior for the chosen theme;

Consistency: Consistent names and signatures for polymorphic operations across classes;

Efficiency: Provide alternative implementations of algorithms that trade time and space;

Extensibility: Must be able to define subclasses for library classes;

Genericity: Should use parameterized class definitions where appropriate.

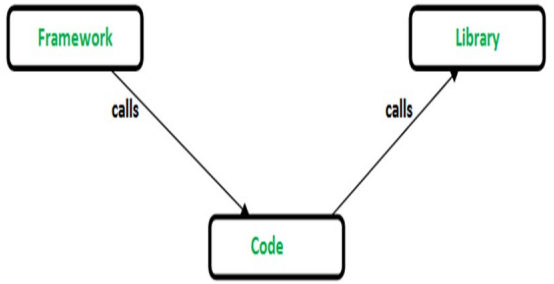
Reuse Plan - Patterns

- A **pattern** is a best practice, a recipe, a time tested solution to a recurring problem that's proved to work and to be the best one among to choose.
- There are patterns for **analysis, architecture, design** and **implementation**.
- A *pattern* comes with **guidelines on when to use it,** as well as **trade-offs on its use.**
- Typically a small number of classes and relationships.
- Advantage is that pattern has been carefully considered by others and has already been applied to past problems successfully.
- **Software Architecture Patterns**: Layered Pattern, Client-Server Pattern
- **Design Patterns**: Gang of Four(GOF). Broken into three categories – **Creational Patterns** for the creation of objects, **Structural Patterns** to provide relationships between

Object Oriented Analysis and Design

Reuse Plan - Frameworks

- Frameworks provide a skeletal structure or ready-made architecture for our application.
- This skeletal structure that is provided must be elaborated to build complete application.
- This elaboration consists of specializing abstract classes with behavior specific to an individual application.
- Consists of more than just th and includes a paradigm for flow of control and shared var



Library Vs Framework

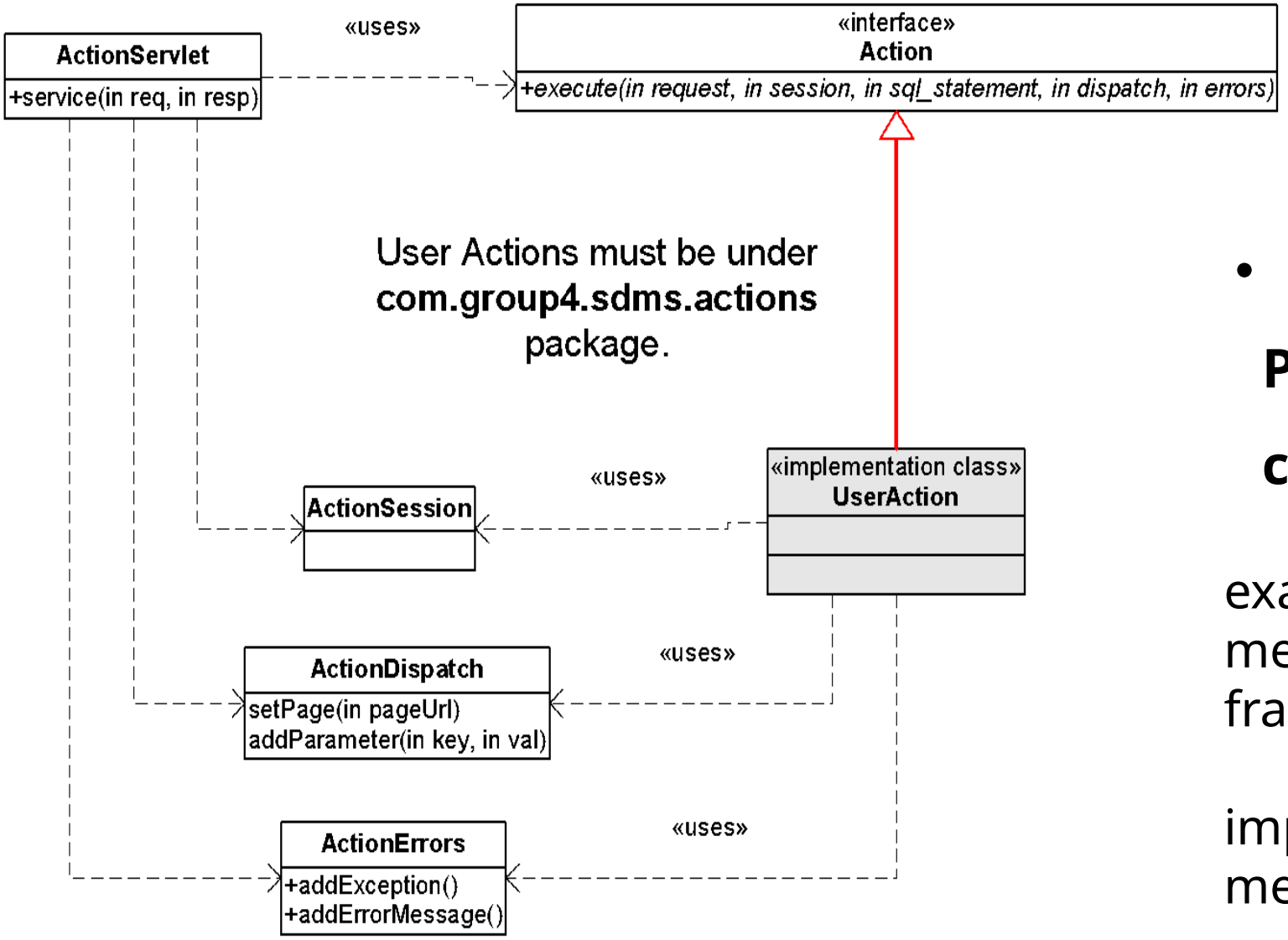
Frameworks in detail

- Pre-written code used by Java developers to develop Java applications or web applications.
- A set of cooperating classes that make up a reusable design for a specific class of software.
- It acts like a skeleton that helps the developer to develop an application by writing their own - The framework is in **control of the programmer**.
- Provides architectural guidance by **partitioning the design into abstract classes.**
- A developer will normally customize a framework to a specific application by

Frameworks in Java

- **Collections:** Provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).
- **Swing:** The part of JFC (Java Foundation Classes) built on the top of AWT and written entirely in Java. The javax.swing API provides all the component classes like JButton, JTextField, JCheckbox, JMenu, etc.
- **AWT:** An abstract window toolkit that provides various component classes like Label, Button, TextField, etc., to show window components on the screen. All these classes are part of the Java.awt package.
- **Spring, Hibernate, Grails, Play, JavaServer Faces (JSF), Google Web**

Framework Example



- Relies on the **'Hollywood Principle' – 'don't call us, we'll call you."**

User-defined classes (for example new subclasses) will receive messages from the predefined framework classes.

Usually handled by implementing super class abstract methods.

- Frameworks **emphasize**

References

- Object - Oriented Modeling and Design With UML by RUMBAUGH and BLAHA,

  Chapter 1 and 14

  - Applying UML and Patterns by Craig Larman, Chapter-34
  - https://www.javatpoint.com/what-is-framework-in-java
  - What is Object-Oriented Modeling (OOM)? - Definition from Techopedia
  - Object oriented methodology (beingintelligent.com)
  - 10 of the Most Popular Java Frameworks of 2020 (stackify.com)

MCQ

1. **What is the primary focus of the analysis model?**
   A. How the system will be implemented
   B. Performance optimization
   C. What the desired system must do
   D. Database design

2. **Which stage involves understanding incomplete or incorrect problem statements?**
   A. System design
   B. Implementation
   C. Analysis
   D. Class design

MCQ

3. **What is the main goal of system design?**
   A. Writing source code
   B. Creating test cases
   C. Devising a high-level system architecture
   D. Drawing UML class diagrams
4. **What is the focus of class design?**
   A. User requirements
   B. Data structures and algorithms
   C. Business workflow
   D. System feasibility

MCQ SOLUTION

1.C
2.C
3.C
4.B

THANK YOU

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Object Oriented Analysis and Design

UE23CS352B

Prof. Shridevi A Sawant

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

PES UNIVERSITY
CELEBRATING 50 YEARS

UE23CS352B: Object Oriented Analysis and Design

Architectural Patterns

Prof. Shridevi A Sawant
Department of Computer Science and Engineering

Acknowledgements:
Prof. Mahitha G & Bhargavi M
Department of Computer Science and Engineering

Architectural Patterns - Agenda

- Introduction – Architectural Patterns
- Model View Controller [MVC]
- How MVC Works?
- MVC Architecture
- Modifying the MVC Design
- Advantages of MVC
- MVC in Java
- Implementation

Introduction to Architectural patterns

- An architectural pattern is a **general**, **reusable** solution to a commonly occurring problem in software architecture
- Helps define the basic characteristics and behavior of an application
- Some architecture patterns naturally lend themselves toward highly scalable applications, whereas other architecture patterns naturally lend themselves toward applications that are highly agile.
- Knowing the characteristics, strengths, and weaknesses of each architecture pattern is necessary in order to choose the one that meets the specific business needs and goals.
- Addresses various issues in software engineering, such as **computer hardware performance limitations, high availability and minimization of a business risk.**
- Layered, event driven, microkernel, microservices, space based and MVC, etc.

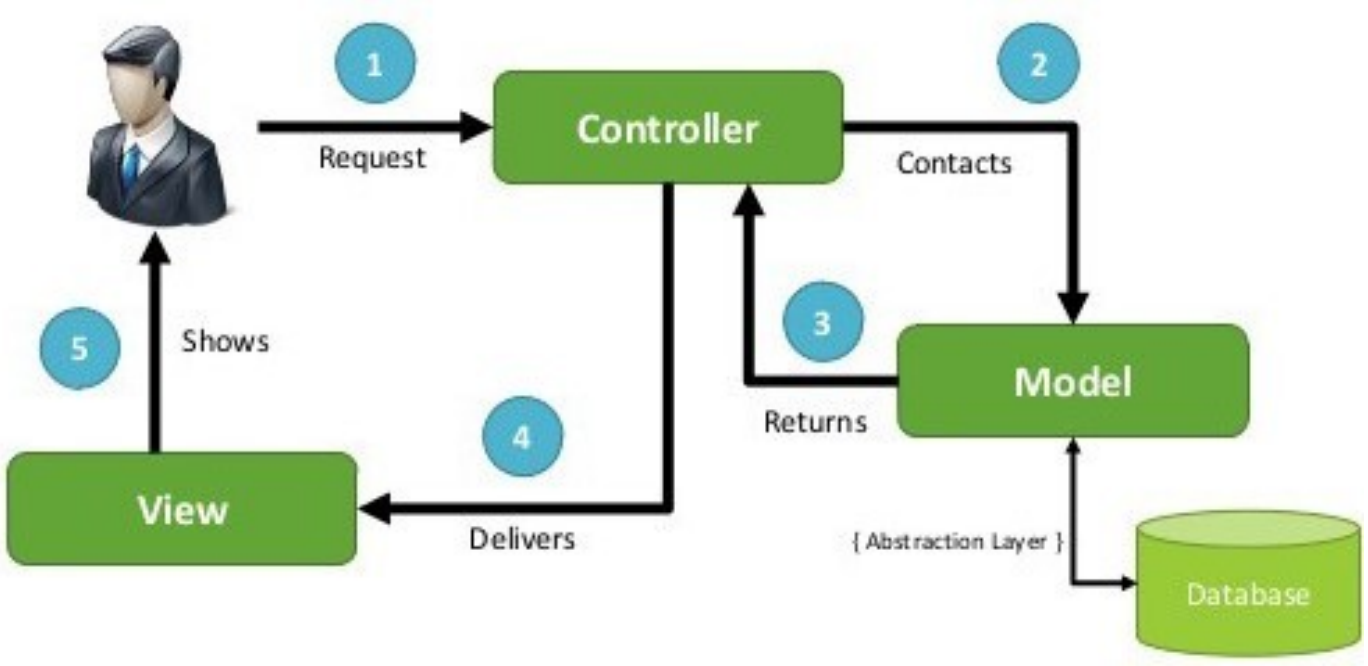
Model View Controller

- MVC - An architectural pattern in Software Engineering.
- First introduced by **Trygve Reenskaug, a Smalltalk developer at the Xerox Palo Alto Research Center in 1979 and** helps to decouple data access and business logic from the manner in which it is displayed to the user.
- A way of designing and building applications that separates application logic from presentation
- Division of application into three main logical components: **model**, **view**, and **controller.**
- A design pattern for computer software considered to distinguish between **the data model, processing control and the user interface.**
- It neatly separates the **graphical interface displayed to the user** from the **code that manages the user actions.**
- Completely separates the calculations and interface from each other

Object Oriented Analysis and Design

How MVC Works?

- Whenever the controller receives a request from the user (either directly or via the view), it puts the model to work. And when the model delivers the data requested in the right format, the controller forwards it to the view

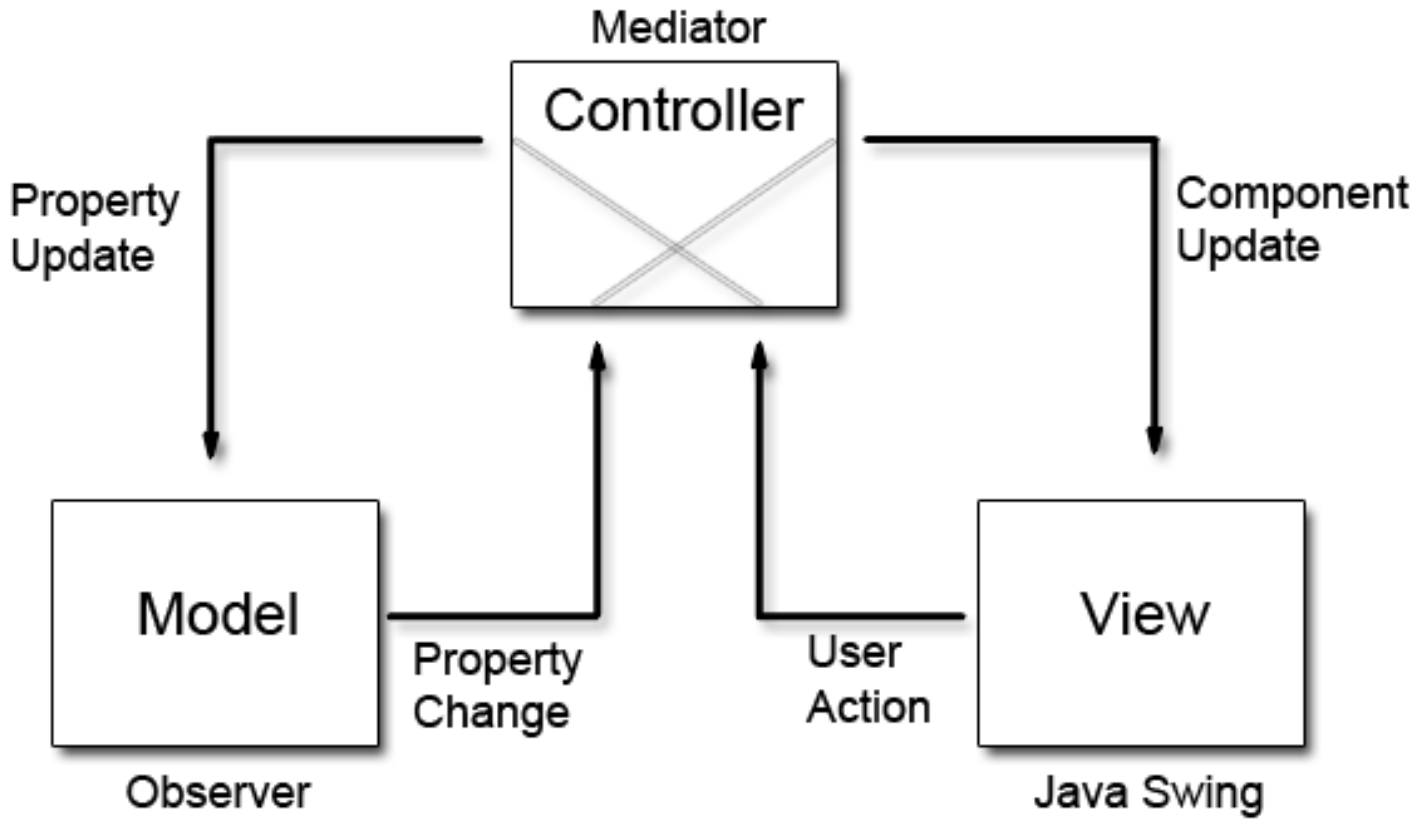


Model View Controller architecture

- Well-known design pattern in the web development field. It is a way to organize t... code.
- Consists of **Data model, presentation information and control information.**
  - **Model:** The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a **software approximation of a real-world process**.
  - **View:** Renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a *push model*, in which the view registers itself with the model for change notifications, or a *pull model*, in which the view is responsible for calling the model when it needs to retrieve the most current data. Represents the presentation layer of application. It is used to visualize the data that the model contains.
  - **Controller:** The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POST HTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user
- The MVC pattern needs all these components to be separated as **different objects.**

Modifying the MVC Design

Program Demo: MVC Employee



Advantages of MVC

- Faster Development Process
- Model can be reused with multiple views
- Less dependency among components – loose coupling
- The modification does not affect the entire model
- Application becomes more understandable
- Not a single massive codebase

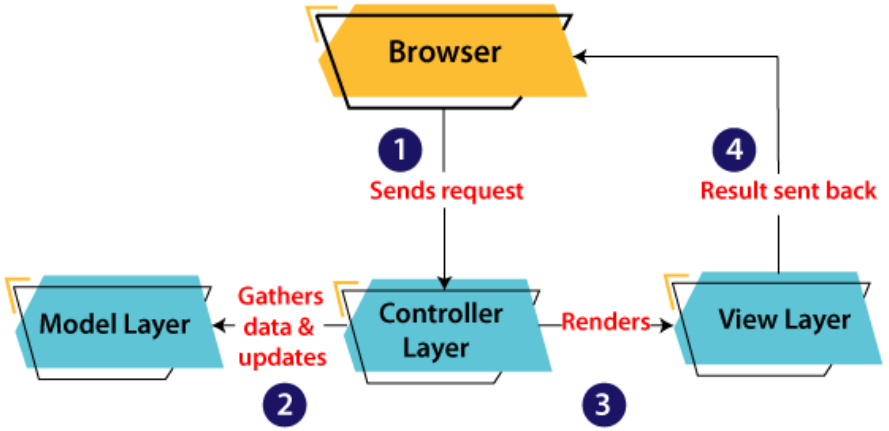
Object Oriented Analysis and Design

Model View Controller architecture in Java

- The Model contains the simple Java classes

  The View used to display the data

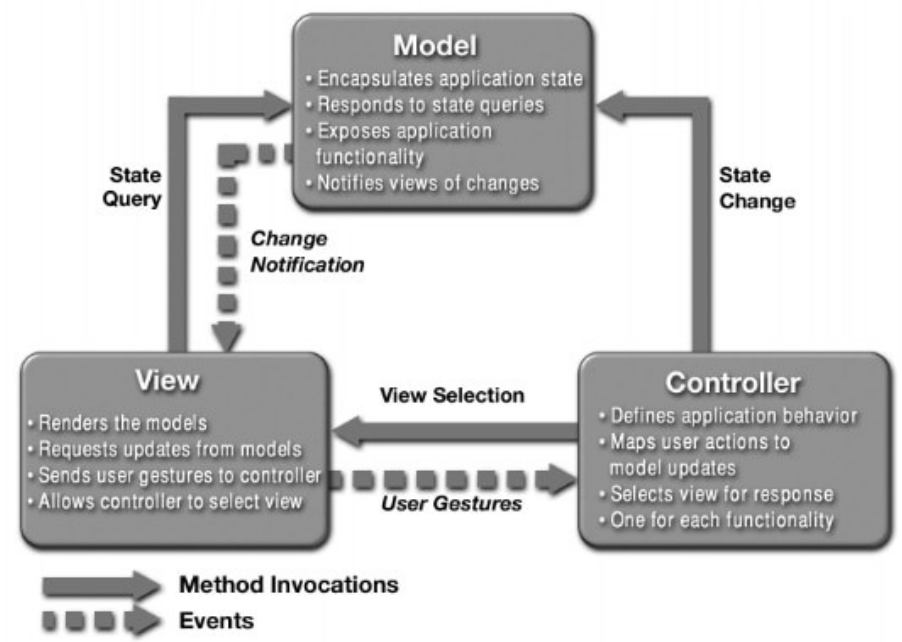
  The Controller contains the servlets/ all en



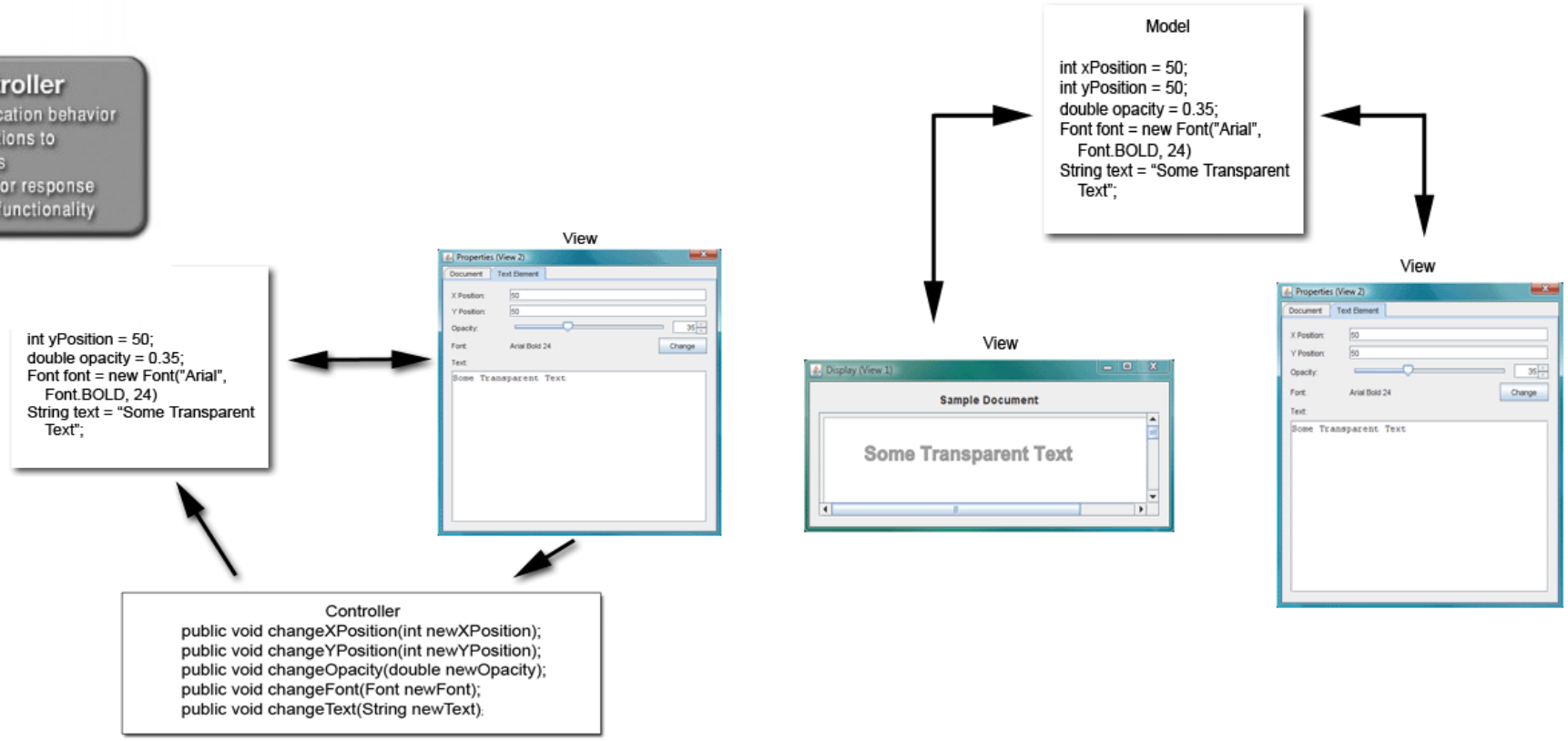
- Due to this separation the user requests are processed as follows:
  - A client (browser) sends a request to the controller on the server side, for a page.
  - The controller then calls the model. It gathers the requested data.
  - Then the controller transfers the data retrieved to the view layer.
  - Now the result is sent back to the browser (client) by the view

Object Oriented Analysis and Design

Model View Controller Implementation



- One of the consequences of this powerful design is that the many views can have the same underlying model.



Implementation of MVC in Java

- **Employee Class, will act as model layer:**

  It represents the business logic for application and also the state of application. The model object fetch and store the model state in the database. Using the model layer, rules are applied to the data that represents the concepts of application.

- **EmployeeView Class, will act as a view layer:**

  View represents the visualization of data received from the model. The view layer consists of output of application or user interface. It sends the requested data to the client, that is fetched from model layer by controller

- **EmployeeContoller Class, will act a controller layer:**

  The controller layer gets the user requests from the view layer and processes them, with the necessary validations. It acts as an interface between Model and View. The requests are then sent to model for data processing. Once they are processed, the data is sent back to the controller and then displayed on the view.

References

- Difference Between Architectural Style, Architectural Patterns and Design Patterns – GeeksforGeeks
- Software Architecture Patterns (oreilly.com)
- MVC Architecture in Java – Javatpoint
- Java SE Application Design With MVC (oracle.com)
- Advantages of using MVC model for effective web application development (brainvire.com)
- MVC Architecture in Java: How to implement MVC in Java? Edureka

MCQ

1. **What is the primary goal of the MVC pattern?**
   A. Improve execution speed
   B. Separate application logic from presentation
   C. Reduce memory usage
   D. Avoid inheritance
2. **What is the responsibility of the Model in MVC?**
   A. Handle user requests
   B. Render the user interface
   C. Represent data and business rules
   D. Control navigation

MCQ

3. **What happens when the controller receives a user request?**
   A. It directly updates the view
   B. It ignores the model
   C. It terminates the request
   D. It puts the model to work and forwards data to the view
4. **Which is an advantage of the MVC pattern?**
   A. Tight coupling
   B. Single massive codebase
   C. Reuse of model with multiple views
   D. Dependency on presentation layer
5. **In the MVC request flow, which component sends the final response back to the client (browser)?**
   A. Model
   B. Controller
   C. View
   D. Client itself

MCQ SOLUTION

1.B
2.C
3.D
4.C
5.C

THANK YOU

Prof. Shridevi A Sawant

Department of Computer Science and Engineering