



Object Oriented Analysis and Design using Java(UE21CS352B)

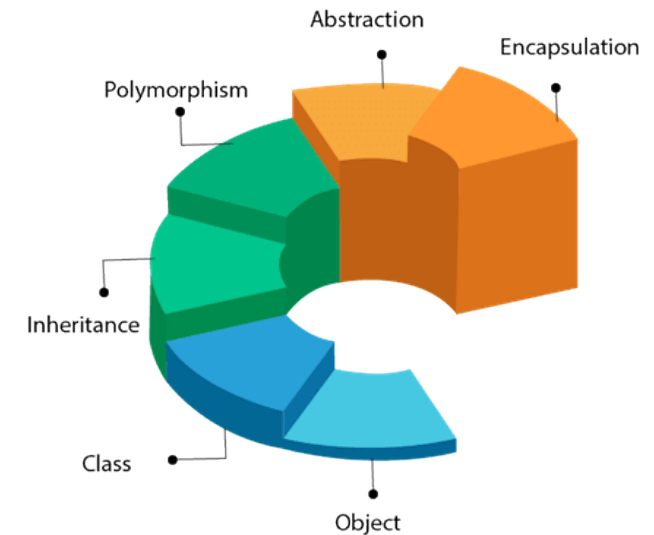
Prof: Mahitha G and Bhargavi M

Department of Computer Science and
Engineering

Object Oriented Analysis and Design using Java

Introduction to OO Programming

OOPs (Object-Oriented Programming System)



Prof : Mahitha G and Bhargavi M

Department of Computer Science and Engineering

Constructor

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

Object Oriented Analysis and Design using Java

Object Oriented Programming: Constructor

Class Student

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

Next we create an object of the above class.

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

Constructors

Default constructor :

- A constructor that has no parameters. If we don't define a constructor for a class, then compiler creates a default constructor.
- Default constructor provides default values to the objects like 0, false, null etc depending on the data type of the instance variables.

Parameterized constructor:

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

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

Example programs: CttDemo.java and CttDemo1.java

Object Oriented Analysis and Design using Java

Object Oriented Programming: Access Modifiers -

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

r.rect()--cannot explicitly invoke constructor

Object Oriented Analysis and Design using Java

Object Oriented Programming: Access Modifiers -

Example

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

By default the access modifier is “default”

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

- 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.

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
- This method is deprecated now



THANK YOU

Mahitha G and Bhargavi M

Department of Computer Science and
Engineering
manithag@pes.edu



Object Oriented Analysis and Design with Java

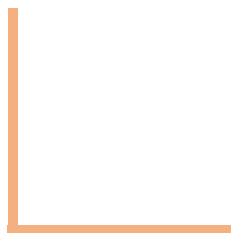
UE21CS352B

Prof. Mahitha G & Bhargavi M

Department of Computer Science and
Engineering

UE20CS352: Object Oriented Analysis and Design with Java

OO Development Process, System Design and Frameworks



Object Oriented Analysis and Design with Java

Agenda

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

Object Oriented Analysis and Design with Java

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 Analysis and Design with Java

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?

Object Oriented Analysis and Design with Java

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.



Object Oriented Analysis and Design with Java

Stages of Object Oriented Methodology



contd..

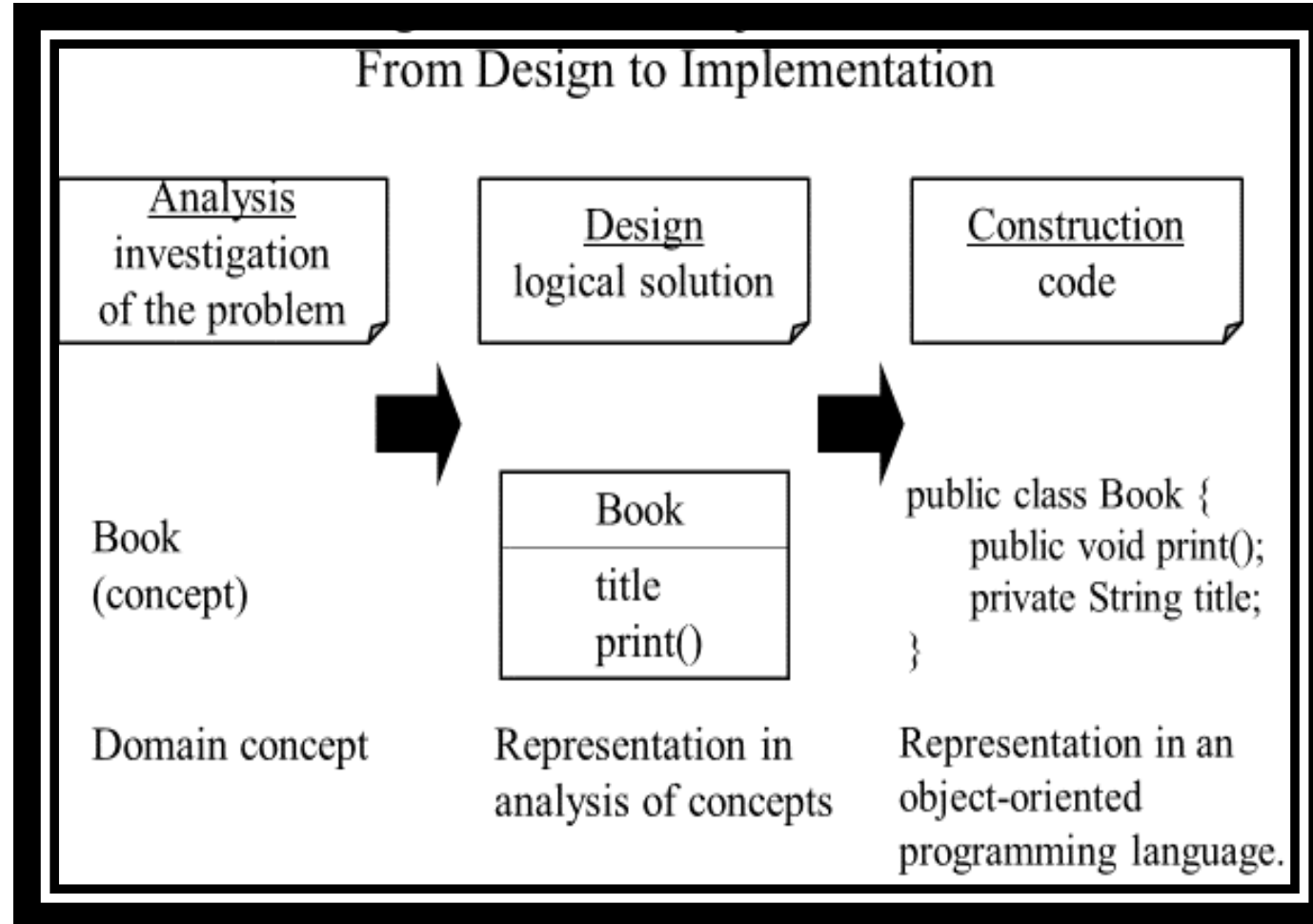
- **System design:** The development teams devise a **high – level strategy** concerning the **system**

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 system remains **flexible and**

Object Oriented Analysis and Design with Java

Sample



Object Oriented Analysis and Design with Java

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 with Java

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

Object Oriented Analysis and Design with Java

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.
- Count the no. of customers, estimate amount of data for each one and multiply i.e. $40 * 20 = 800$

Object Oriented Analysis and Design with Java

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.

Object Oriented Analysis and Design with Java

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.
- Performs a set of specific and well-defined operations. Examples :

Object Oriented Analysis and Design with Java

Reuse Plan - Patterns



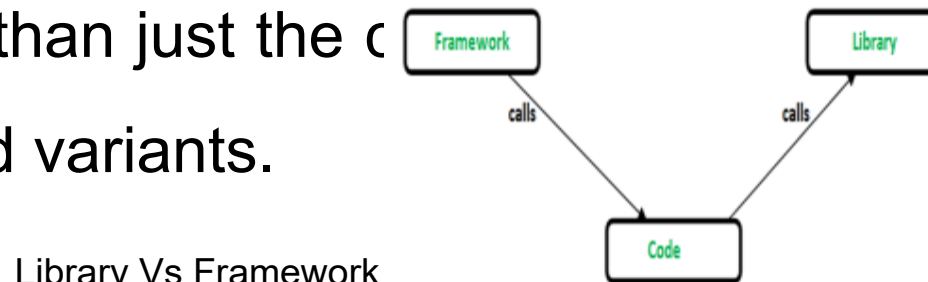
- A **pattern** is a best practice, a recipe, a time tested solution to a recurring | that's proved to work and to be the best one among to choose.
- There are patterns for **analysis, architecture, design** and **implementation**.
- A *pattern* comes with **guidelines on when to use it**, as well as **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 objects and

Object Oriented Analysis and Design with Java

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 the c



Library Vs Framework

Object Oriented Analysis and Design with Java

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 “subclassing” and composing instances of framework classes.

Object Oriented Analysis and Design with Java

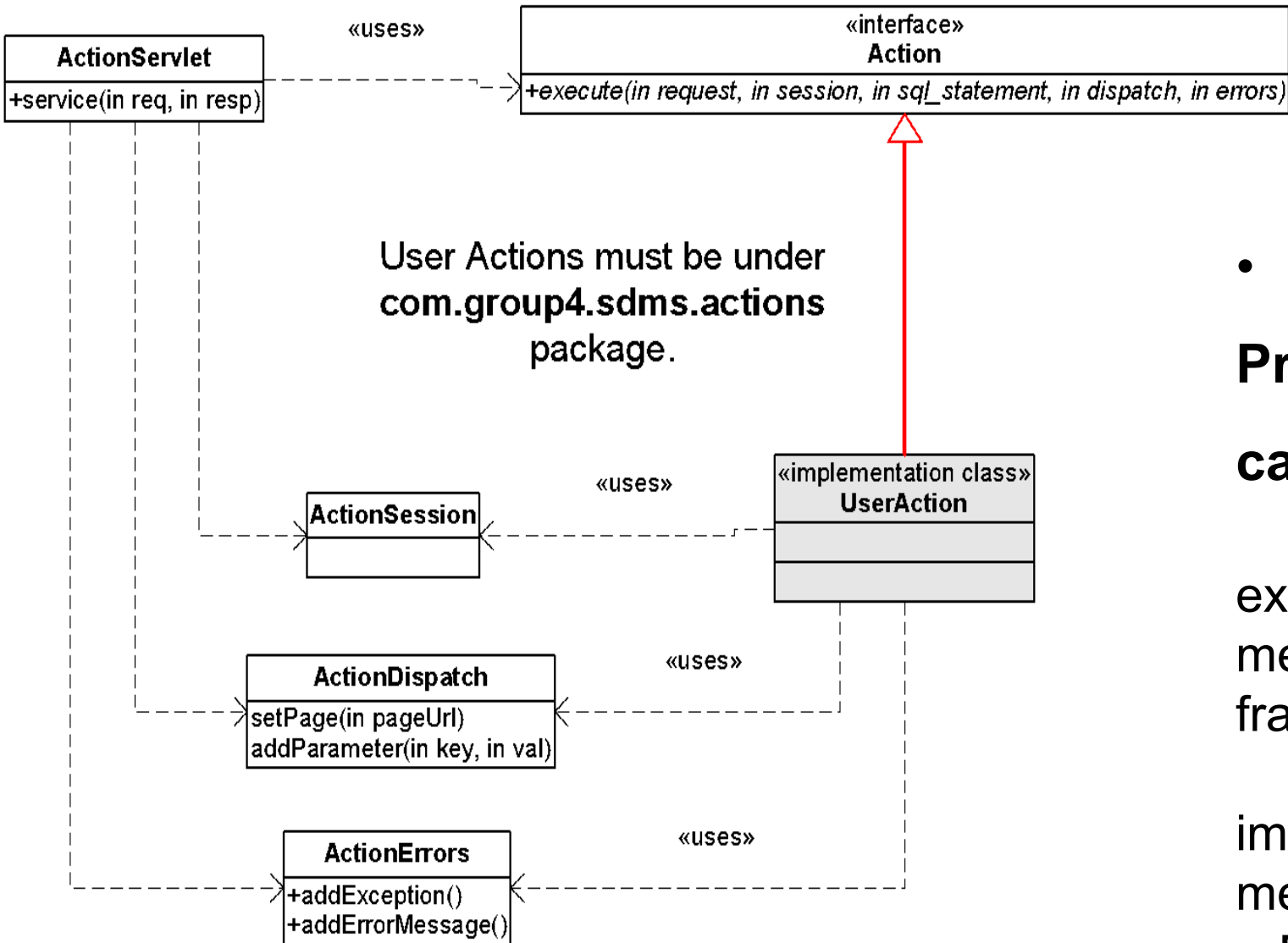
Frameworks in Java



- **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**

Object Oriented Analysis and Design with Java

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 design**

Object Oriented Analysis and Design with Java

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 \(stackify.com\)](#)



THANK YOU

Mahitha G & Bhargavi M
Department of Computer Science and
Engineering



Object Oriented Analysis and Design with Java - UE21CS352B

Mahitha G and Bhargavi M

Department of Computer Science and Engineering

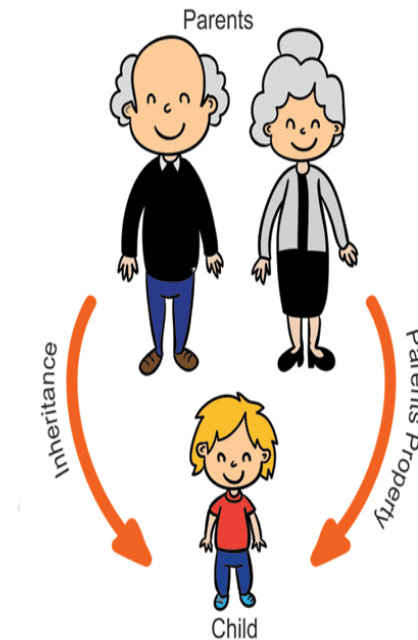
Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission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

UE19CS353: Object Oriented Analysis and Design with Java

Interfaces

Mahitha G and Bhargavi M

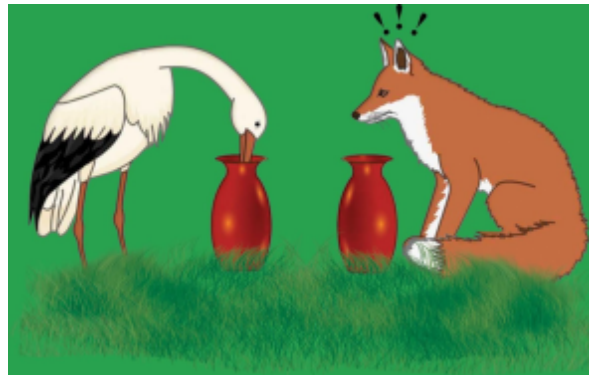
Department of Computer Science and Engineering



Object Oriented Analysis and Design with Java

Interface in Java

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



So next
topic is
interfaces
in java

Object Oriented Analysis and Design with Java

Interface in Java



Syllabus of OOPJ is an interface. The teachers implement this interface. You are the clients. But unfortunately, in this case, you can not chose the implementation! The students of A & G section are tied to my implementation! In our department, we have already experimented with students choosing the implementation – choose which teacher (elective)! A day may come when you can choose a different teacher for each topic!

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

```
public interface Displayable
{
    void disp();
}
```

Object Oriented Analysis and Design with Java

Interface in Java

- We know, objects define their interaction with the outside world through the methods that they expose.
 - Methods form the object's interface with the outside world;
 - Buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing.
 - You press the "power" button to turn the television on and off.
- In its most common form, an interface is a group of related methods with empty bodies.

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

```
interface Bicycle {  
  
    // wheel revolutions per minute  
    void changeCadence(int newValue);  
  
    void changeGear(int newValue);  
  
    void speedUp(int increment);  
  
    void applyBrakes(int decrement);  
}
```

Object Oriented Analysis and Design with Java

Interface in Java



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

```
class AAABicycle implements Bicycle {
```

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

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

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

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

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

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

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

Object Oriented Analysis and Design with Java



Interface in Java

Interfaces can be extended

```
Interface i1
{
    Void display();
}
```

```
Interface i2 extends i1
{
    Void print();
}
```

```
Class sample implements i2
{
```

```
//should override both display and print
}
```

Object Oriented Analysis and Design with Java

Interface in Java



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

Object Oriented Analysis and Design with Java

Inheritance Vs Interface



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

Object Oriented Analysis and Design with Java

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 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.

Object Oriented Analysis and Design with Java

Interface in Java



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

- Definitely NO.

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

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

- **Can an interface extend an interface?** ◦

Definitely YES.

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

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

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

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

Object Oriented Analysis and Design with Java

Interface in Java

Programs to demonstrate Inheritance and interface

Interfacedemo.java

Interfacedemo1.java

Interfacedemo2.java

interfacedemo_new.java





THANK YOU

Mahitha G and Bhargavi M

Department of Computer Science and
Engineering
manithag@pes.edu



Object Oriented Analysis and Design with Java - UE21CS352B

Mahitha G and Bhargavi M

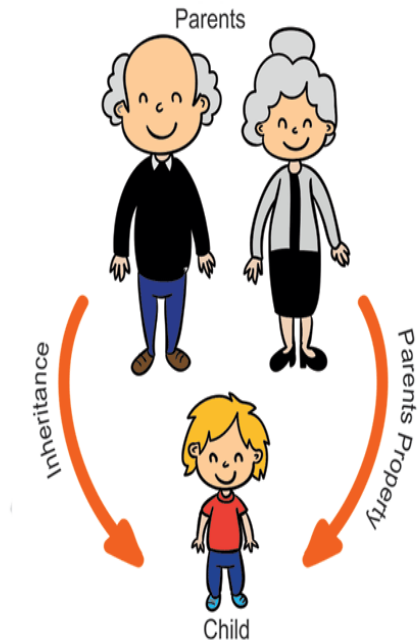
Department of Computer Science and Engineering

UE19CS353: Object Oriented Analysis and Design with Java

Object Oriented Concepts - Inheritance

Mahitha G and Bhargavi M

Department of Computer Science and Engineering



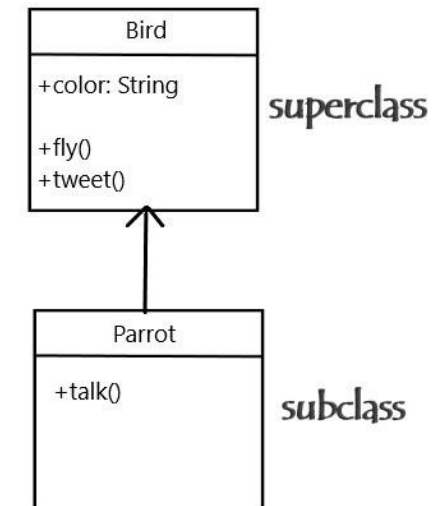
Object Oriented Analysis and Design with Java

Inheritance

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



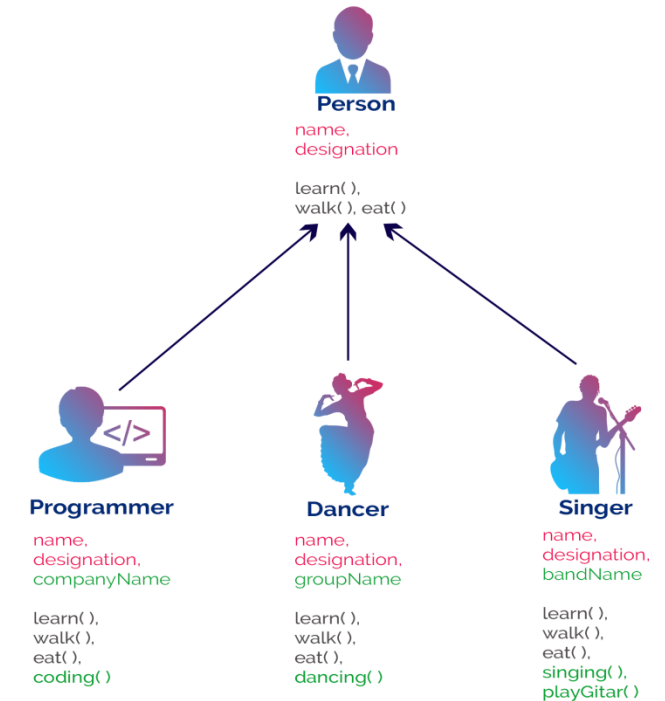
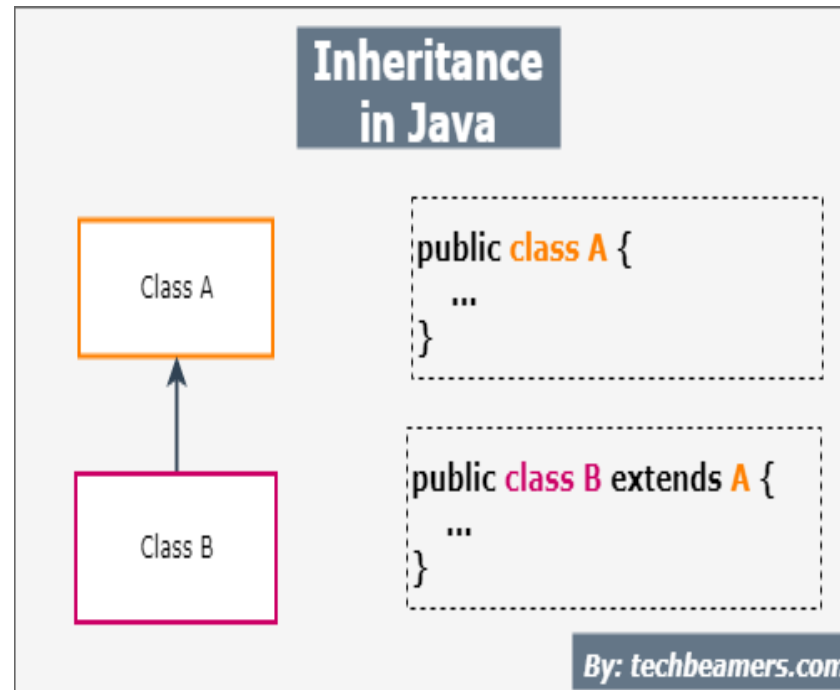
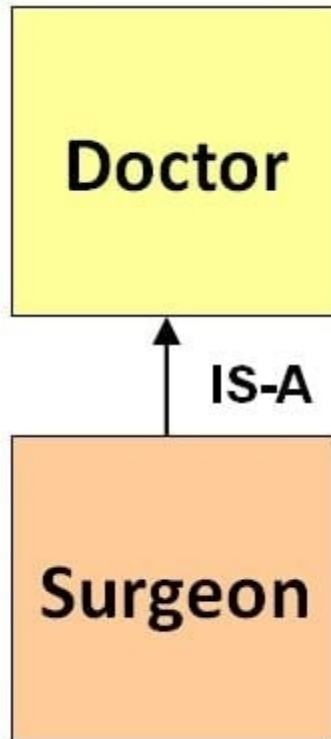
Inheritance



Object Oriented Analysis and Design with Java

Inheritance

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



Object Oriented Analysis and Design with Java

Inheritance



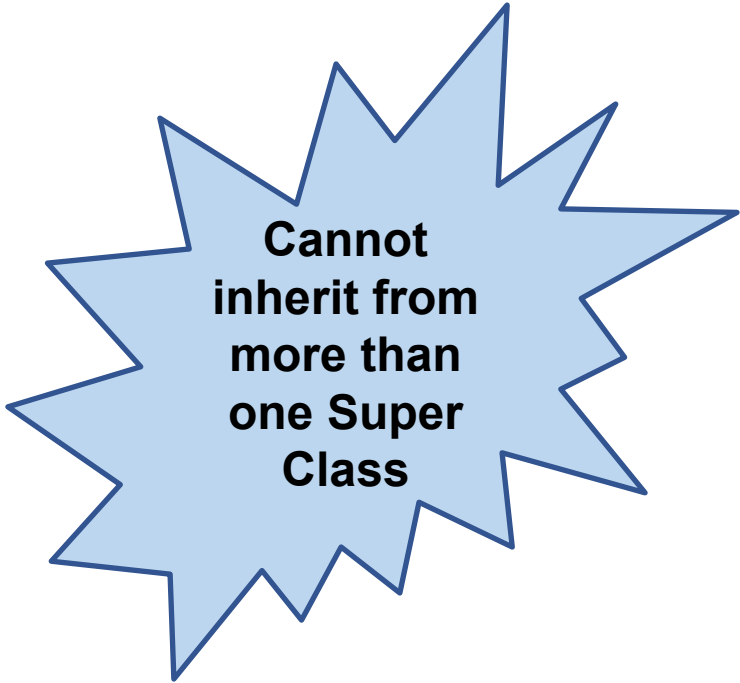
Syntax:

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

Object Oriented Analysis and Design with Java

Inheritance in Java

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



**Cannot
inherit from
more than
one Super
Class**

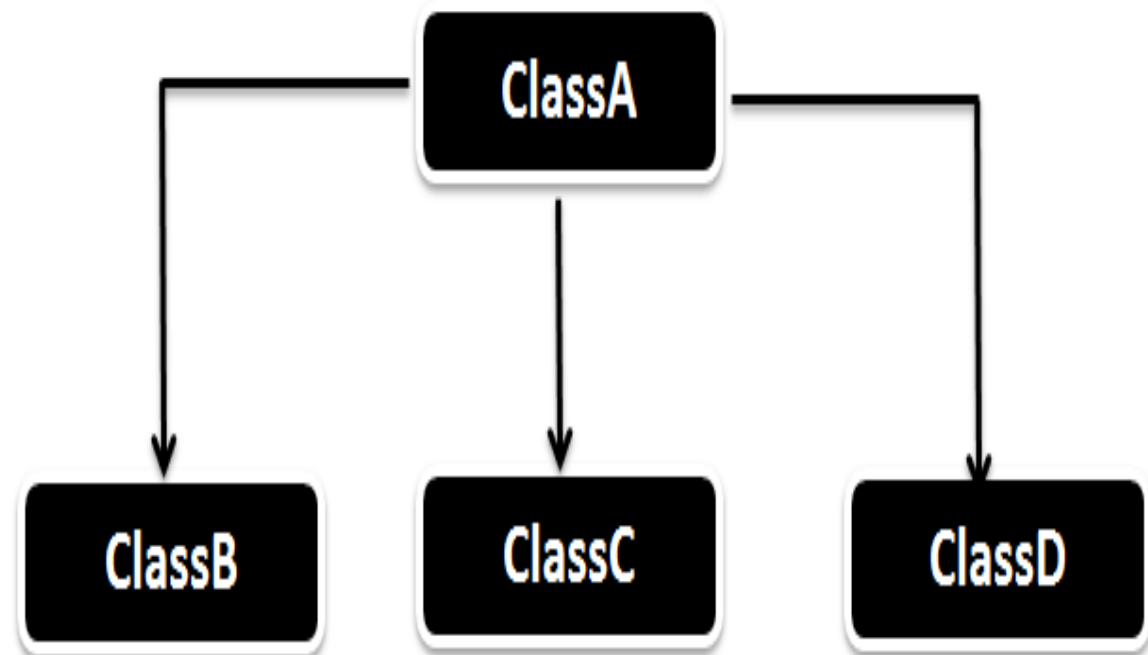
Object Oriented Analysis and Design with Java

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.

- Single Level Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance



Object Oriented Analysis and Design with 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);
    }
}
```

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();
    }
}
```

Multilevel Inheritance

Programming Example using instanceof keyword

Multileveldemo.java

In Java, instanceof is a keyword used for checking if a reference variable contains a given type of object reference or not. Following is a Java program to show different behaviors of instanceof. Henceforth it is known as a comparison operator where the instance is getting compared to type returning boolean true or false as in Java we do not have 0 and 1 boolean return types.

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
    }
}
```

Object Oriented Analysis and Design with Java

Access Specifier



- Access specifiers in Java control the visibility and accessibility of classes, methods and variables within a program.
- They ensure encapsulation and maintain security and integrity of code.
- Programming example : p1 and p2 folders.

	Default	Private	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package, subclass	Yes	No	Yes	Yes
Same package, non-subclass	Yes	No	Yes	Yes
Different package, subclass	No	No	Yes	Yes
Different	No	No	No	Yes

Object Oriented Analysis and Design with Java

Access Specifier – default



```
package p1;

class DefaultExample // default access specifier
{
    void display()
    {
        System.out.println("Hello World!");
    }
}

package p2;
import p1.*;
class DefaultExample2 // default access specifier
{
    public static void main(String args[])
    {
        DefaultExample obj = new DefaultExample();
        obj.display();
    }
}
```

- When no access specifier is specified for a class, method or data member – it is said to have the default access specifier.
- Data members, classes or methods that are not declared using any access specifiers are accessible only within the same package.

Object Oriented Analysis and Design with Java

Access Specifier – private

```
package p1;
class A
{
    private void display()
    {
        System.out.println("Private Method");
    }
}
class B
{
    public static void main(String args[])
    {
        A obj = new A();
        obj.display(); // trying to access private method of another class
    }
}
error: display() has private access in A
    obj.display();
```

- Any other class of the same package will not be able to access these members.
- The methods or data members declared as private are accessible only within the class in which they are declared.

Object Oriented Analysis and Design with Java

Access Specifier – public



```
package p1;
public class A
{
    public void display()
    {
        System.out.println("Public method");
    }
}

package p2;
import p1.*;
class B {
    public static void main(String args[])
    {
        A obj = new A();
        obj.display();
    }
}
```

Output - Public method

- **Public access specifier has the widest scope among all other access specifiers.**
- **Classes, methods, data members that are declared public are accessible from everywhere in the program. There is no restriction on the scope of the data members.**

Object Oriented Analysis and Design with Java

Access Specifier – protected



```
package p1;
public class A
{
    protected void display()
    {
        System.out.println("Protected method");
    }
}
package p2;
import p1.*; // importing all classes in package p1
class B extends A // Class B is subclass of A
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.display();
    }
}
```

Output - Protected method

- **Methods or data members declared as protected are accessible within the same package or subclasses in different packages.**
- **Facilitates inheritance and code organization.**

Object Oriented Analysis and Design with Java

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).

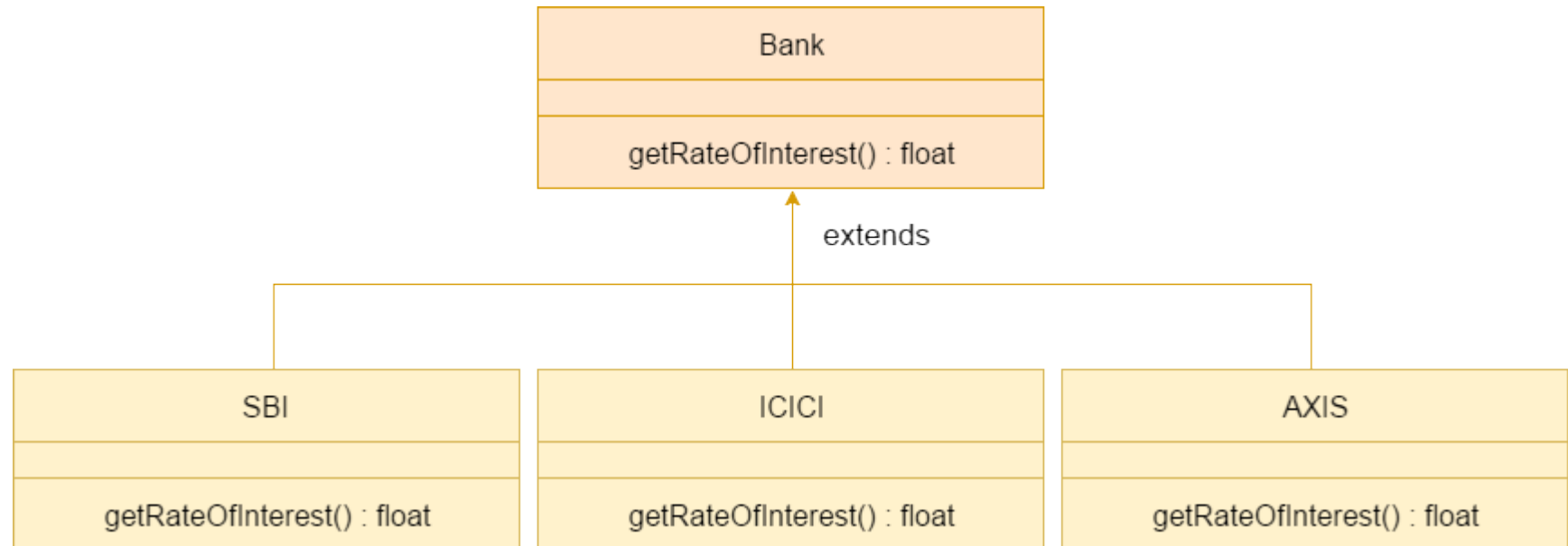
Object Oriented Analysis and Design with Java

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.



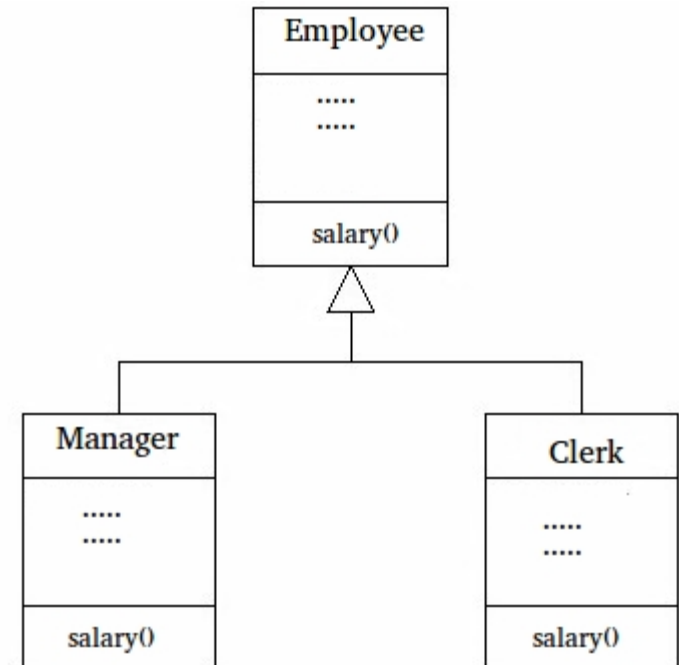
Object Oriented Analysis and Design with Java

Method Over-riding - Example program



An Example of Java Method Overriding

Program **override.java**



Object Oriented Analysis and Design with Java

Super keyword in java



Super keyword in java can be used for

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Programming example

superdemo.java

Superdemo1.java

Superdemo2.java



THANK YOU

Mahitha G & Bhargavi M

Department of Computer Science and
Engineering
manithag@pes.edu



Object Oriented Analysis and Design using Java - UE21CS352

Prof. Mahitha G and Bhargavi M

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Single Rooted Hierarchy and Object Class

Acknowledgement: Prof. Priya Badarinath and Prof. Sindhu

Department of Computer Science and Engineering

Single Rooted Hierarchy

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

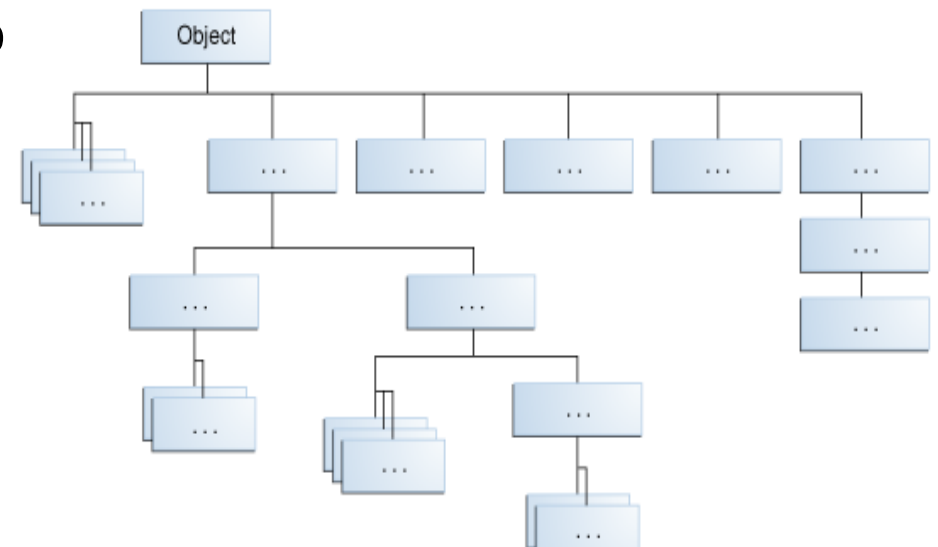
The singly-rooted hierarchy is common with most other object-oriented programming languages.

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

Object 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 o
- This is defined in the **java.lang** package



Object Oriented Analysis and Design using Java

Object class Methods



- **public final Class<?> getClass()**

- Returns the runtime class of this Object. The returned Class object is the object that is locked by static synchronized methods of the represented class.

- **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 Oriented Analysis and Design using Java

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:

- It is **reflexive**: for any non-null reference value x, x.equals(x) should return true.
- It is **symmetric**: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is **transitive**: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is **consistent**: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, x.equals(null) should return false.

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

Object Oriented Analysis and Design using Java

Object class

Programming Example: Obejectdemo.java

Method	Purpose
Object clone()	Creates a new object that is the same as the object being cloned.
boolean equals(Object <i>object</i>)	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 <i>milliseconds</i>) void wait(long <i>milliseconds</i> , int <i>nanoseconds</i>)	Waits on another thread of execution.

Object Oriented Analysis and Design using Java

Object class

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

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

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

Object Oriented Analysis and Design using Java

Object class

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

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

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



THANK YOU

Prof. Mahitha G and Bhargavi M

Department of Computer Science and Engineering

mahithag@pes.edu



Object Oriented Analysis and Design with Java (UE20CS352)

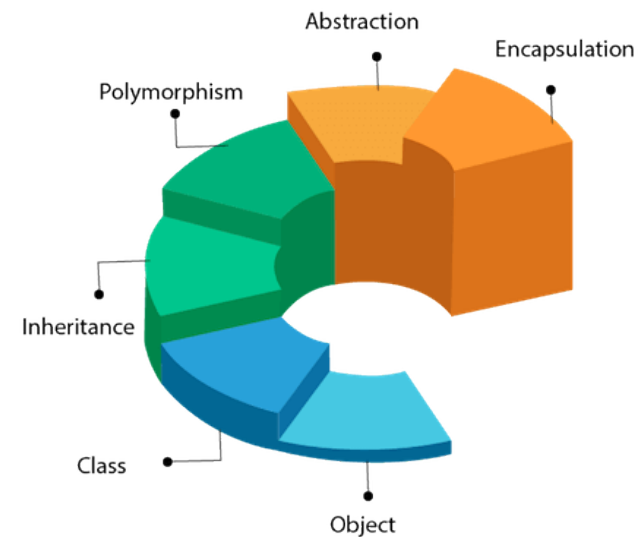
Prof Mahitha G and Bhargavi M
Department of Computer Science and Engineering

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 using Java

Introduction to OO Programming

OOPs (Object-Oriented Programming System)



Mahitha G Bhargavi M

Department of Computer Science and Engineering

There are two types of methods:

Instance Methods

Instance methods are methods which require an object of its class to be created before it can be called. To invoke an 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

Object Oriented Analysis and Design using Java

Method Types



Example to illustrate instance methods:

```
class Sample {  
  
    String name = "";  
  
    public void setname(String s) { name = s; }  
}  
  
class example1 {  
    public static void main(String[] args)  
    {  
  
        // create an instance of the class.  
        Sample obj1 = new Sample();  
  
        // calling an instance method i  
        obj1.setname(" Ramu");  
        System.out.println(obj1.name);  
    }  
}
```

Object Oriented Analysis and Design using Java

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 using Java

Static Variable



Example to illustrate static variable:

TestStaticVariable.java

Example counter without static variable

Counter.java

Example counter with static variable

Counter2.java

Why Java
main
method is
static????

Static Method

- A static method can be invoked without the need for creating an instance of a class.
- Static methods cannot access or change the values of instance variables, but they can access or change the values of static variables.
- Static methods cannot call non-static methods.
- Static methods are associated with the class, not objects of the class.

Object Oriented Analysis and Design using Java

Static Method



Example to illustrate static methods:

```
class Sample {  
  
    public static String name = "";  
  
    public static void setname(String s)  
    {  
        name= s;  
    }  
}  
  
class example2 {  
    public static void main(String[] args)  
    {  
  
        // Accessing the static method setname  
        // and field by class name itself.  
        Sample.setname("abhiram");  
        System.out.println(Sample.name);  
  
        // Accessing the static method setname  
        Sample obj = new Sample();  
        obj.setname("manish");  
        System.out.println(obj.name);    }}
```

Object Oriented Analysis and Design using Java

Static Block



Java static block

Is used to initialize the static data member.

It is executed before the main method at the time of classloading 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

Example: [StaticDemo.java](#)

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

Object Oriented Analysis and Design using Java

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.



THANK YOU

Mahitha G and Bhargavi M

Department of Computer Science and
Engineering
mahithag@pes.edu



CELEBRATING 50 YEARS

Object Oriented Analysis and Design with Java

UE20CS35

2 Prof. Mahitha G & Bhargavi M

Department of Computer Science and Engineering

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

UE20CS352: Object Oriented Analysis and Design with

Java

Architectural Patterns

Ack: Prof Sindhu P

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Architectural Patterns - Agenda

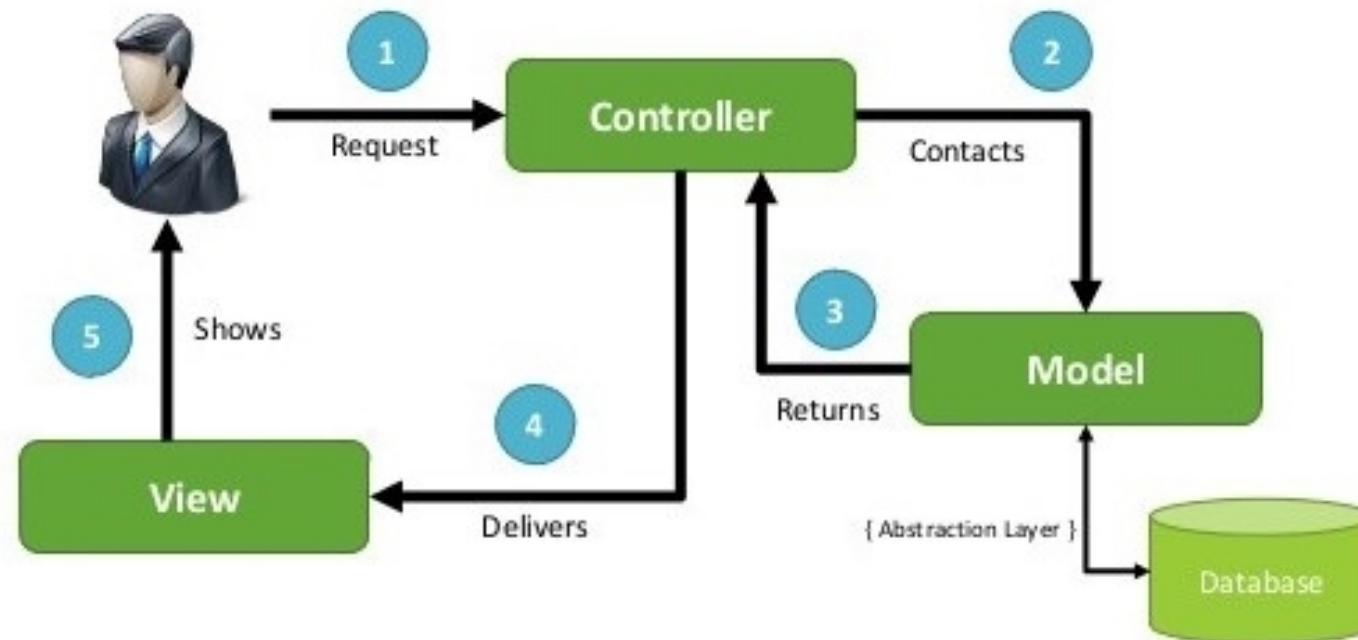
- **Introduction – Architectural Patterns**
- **Model View Controller [MVC]**
- **How MVC Works?**
- **MVC Architecture**
- **Modifying the MVC Design**
- **Advantages of MVC**
- **MVC in Java**
- **Implementation**

- 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.**

- 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

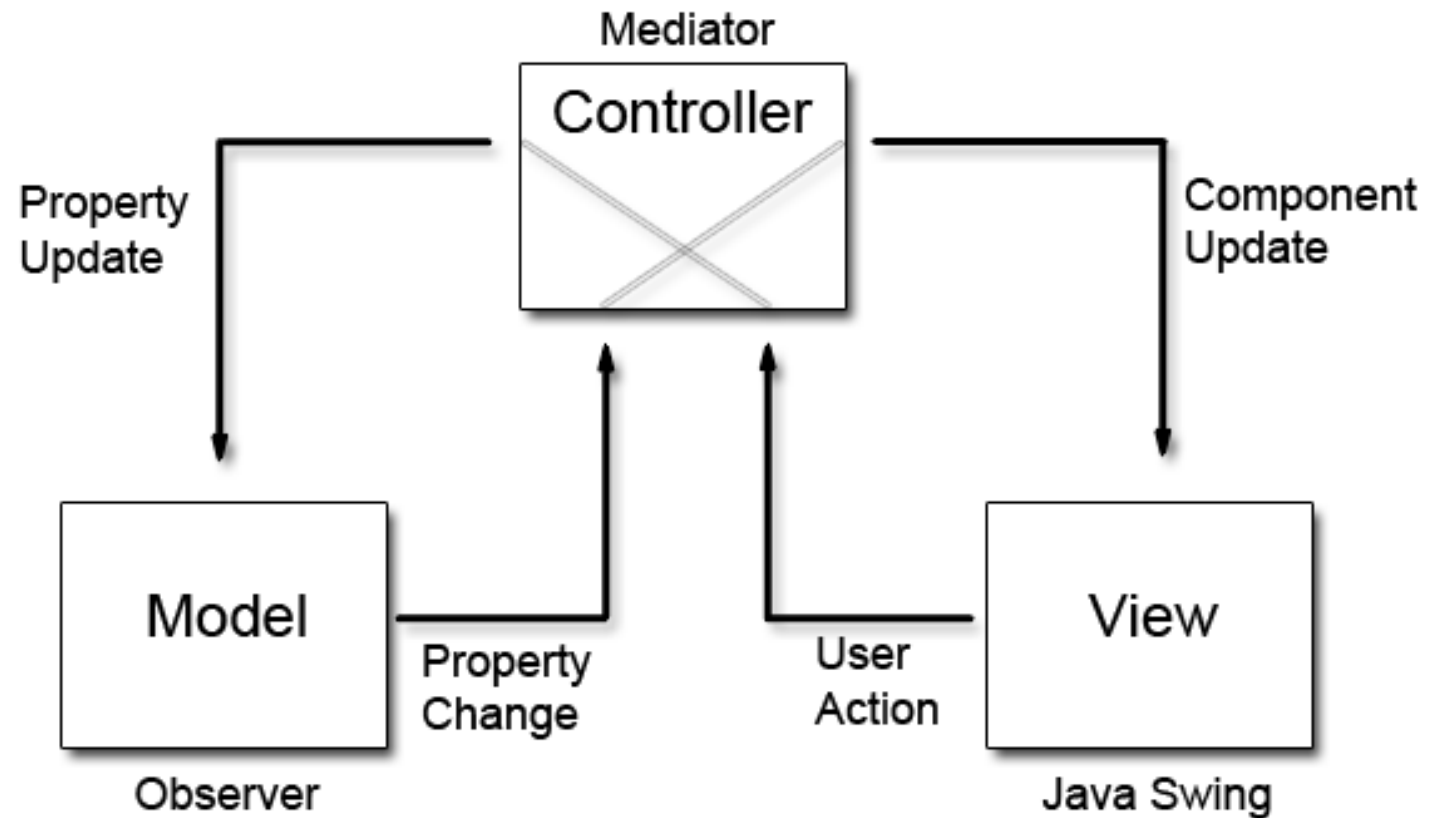
How MVC Works?

- 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, t



- Well-known design pattern in the web development field. It is a way to organize the 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

Program Demo: MVC Employee



Object Oriented Analysis and Design using Java

Advantages of MVC

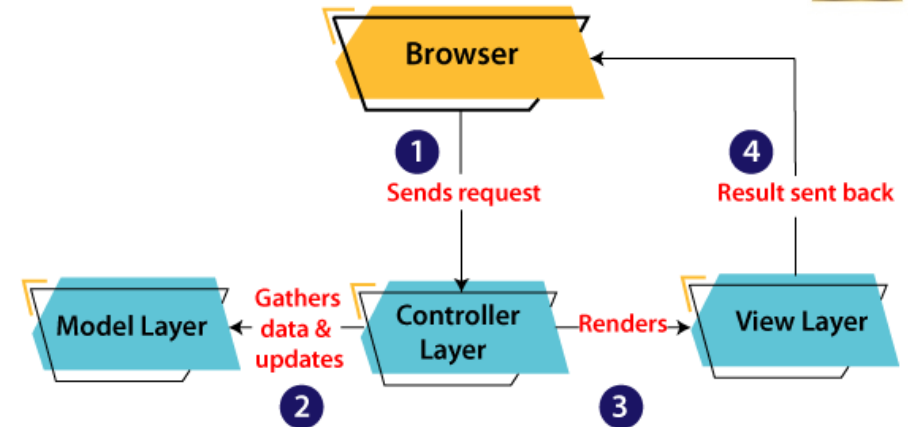


- Faster Development Process
- Model can be reused with multiple views
- Less dependency among components – loose coupling
- The modification does not affect the entire model
- Application becomes more understandable
- Not a single massive codebase

Object Oriented Analysis and Design using Java

Model View Controller architecture in Java

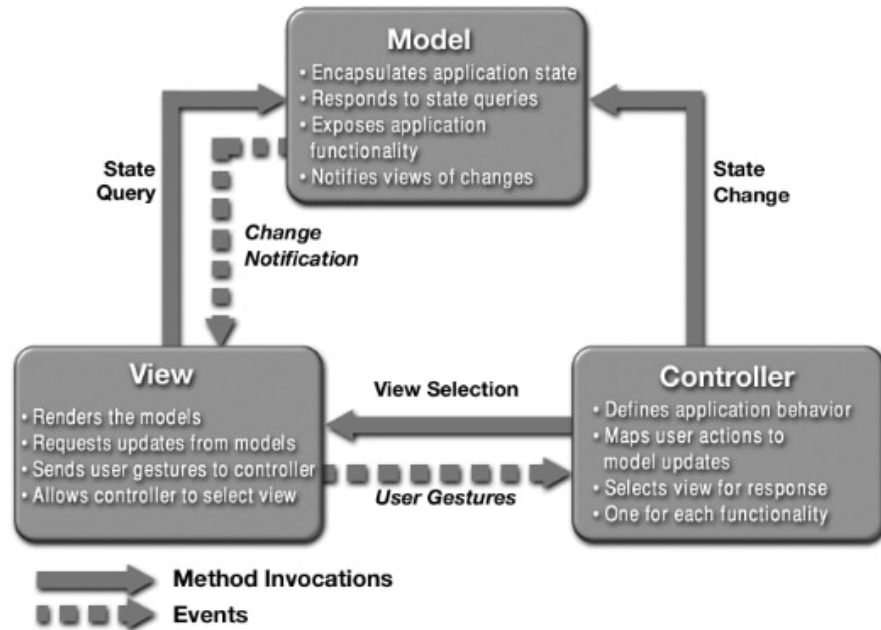
- The Model contains the simple Java classes,
The View used to display the data
The Controller contains the [servlets](#)/ all entities.



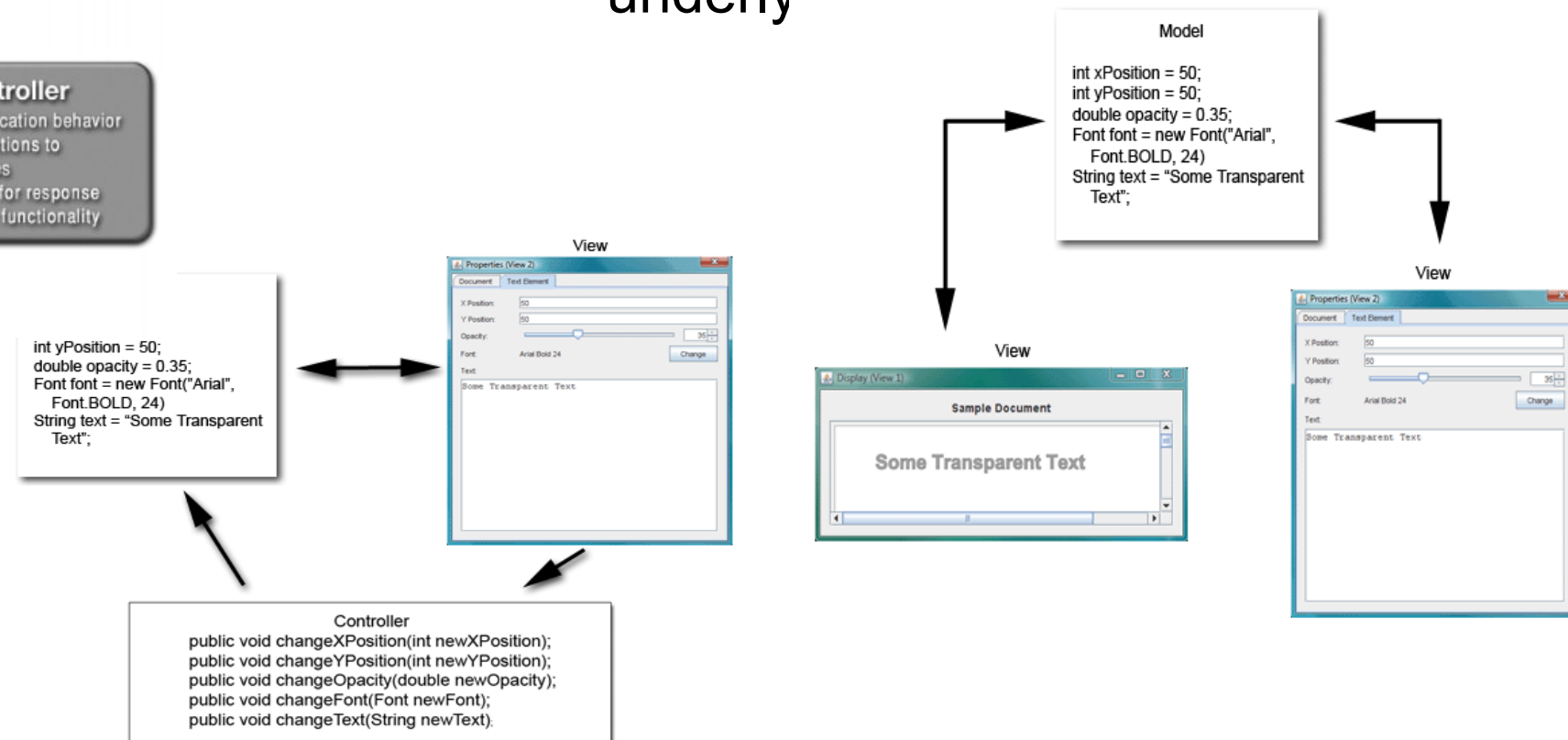
- Due to this separation the user requests are processed as follows:
 - A client (browser) sends a request to the controller on the server side, for a page.
 - The controller then calls the model. It gathers the requested data.
 - Then the controller transfers the data retrieved to the view layer.
 -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

Object Oriented Analysis and Design using Java

Model View Controller Implementation



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



- **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

- **EmployeeController 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.

- [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](#)



THANK YOU

Prof. Mahitha G & Bhargavi M

Department of Computer Science and
Engineering



Object Oriented Analysis and Design with Java - UE21CS352B

Prof. Mahitha G and Prof Bhargavi M

Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

UE20CS352

Parameter Passing – Value Types and Reference Types

Prof. Mahitha G and Bhargavi M

Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

Agenda

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

Object Oriented Analysis and Design with Java

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**

Object Oriented Analysis and Design with Java

Parameter Passing

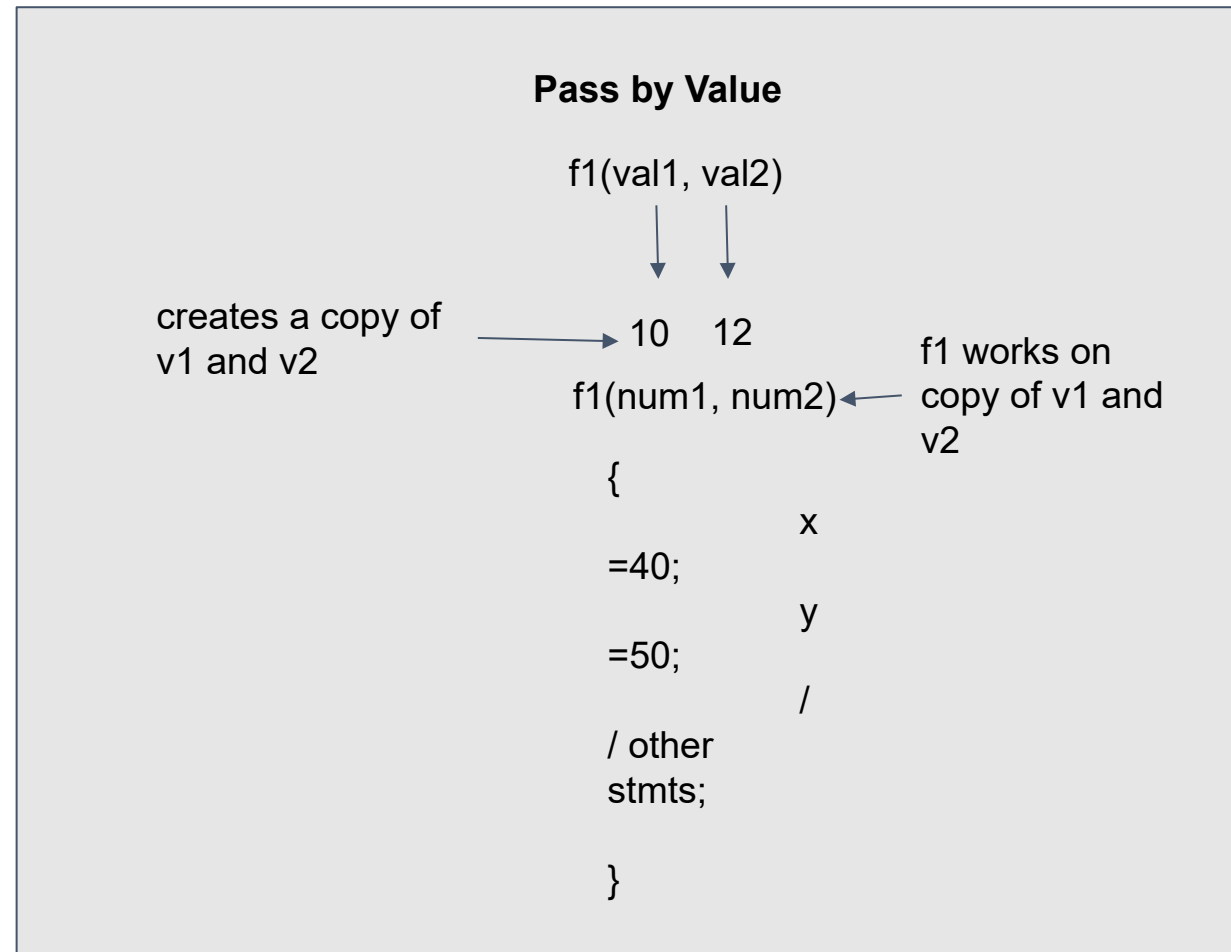
Coding examples – callbyvaluedemo.java

Object Oriented Analysis and Design with Java

Parameter Passing

Value types

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

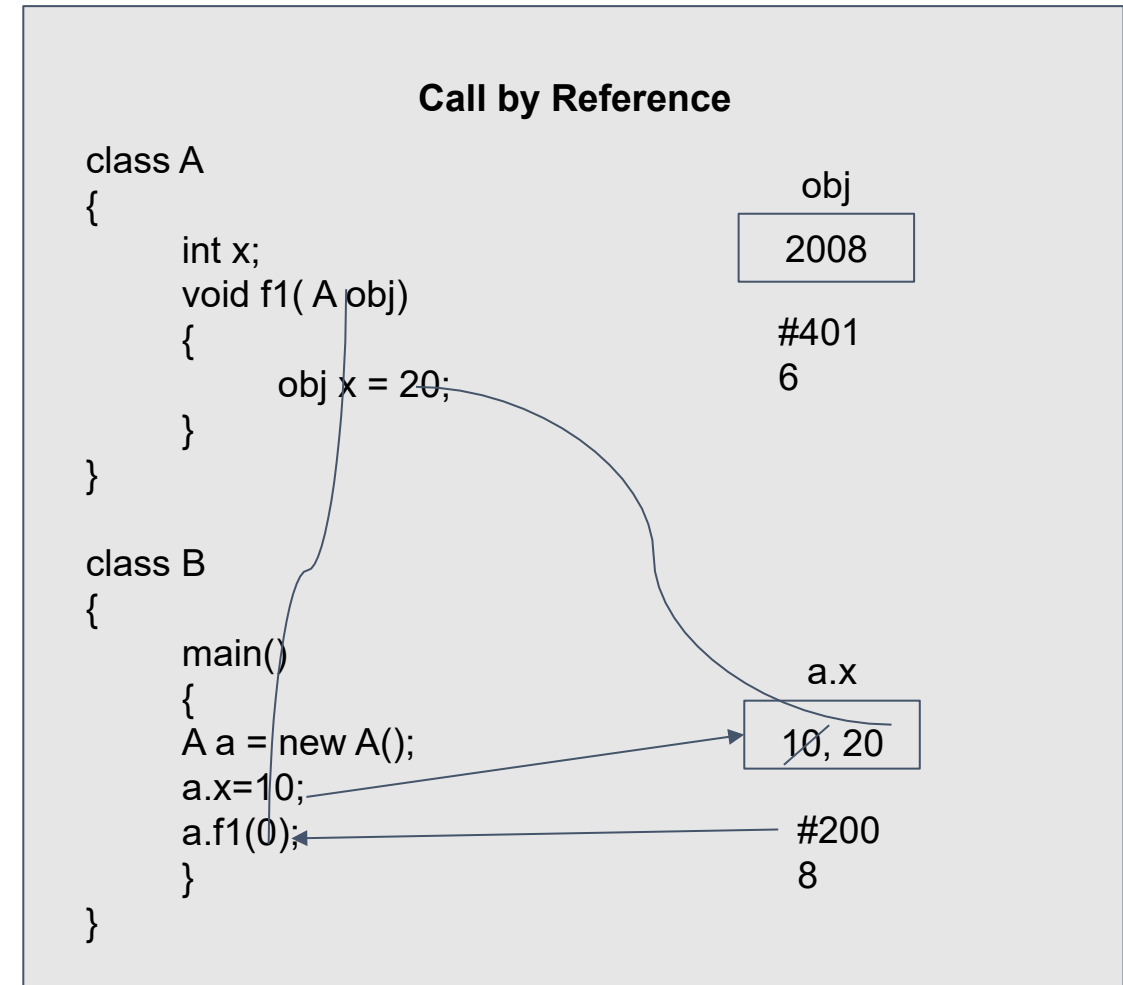


Object Oriented Analysis and Design with Java

Parameter Passing

References- alias

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



Object Oriented Analysis and Design with Java

Parameter Passing

Programming Example: paramdemo.java

Reference types continued..

- **Trick 1:** The changes are not reflected back if we change the object itself to refer some other location or object
- **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 = new Test();      t.x = 10;      }
}
```

```
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;      }
}
```




THANK YOU

Prof Mahitha G and Bhargavi M

Department of Computer Science and Engineering



Object Oriented Analysis and Design using Java - UE21CS352B

Prof. Mahitha G and Bhargavi M

Department of Computer Science and Engineering



PES
UNIVERSITY
CELEBRATING 50 YEARS

Object Oriented Analysis and Design using Java

Abstract Class

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Abstract class and Object class - Agenda



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

Object Oriented Analysis and Design using Java

Abstract class



Introduction

- An abstract class is a class that is declared abstract—it may or may not include abstract methods.
- Abstract classes cannot be instantiated, but they can be subclassed.
- An **abstract method** is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:
 - `abstract void moveTo(double deltaX, double deltaY);`

- If a class includes abstract methods, then the class itself must be declared abstract

```
public abstract class GraphicObject {  
    // declare fields  
    // declare non abstract methods  
    abstract void draw();  
}
```

When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.

Which should you use, abstract classes or interfaces?

- Consider using abstract classes if any of these statements apply to your situation:
 - You want to share code among several closely related classes.
 - You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
 - You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.
- Consider using interfaces if any of these statements apply to your situation:
 - You expect that unrelated classes would implement your interface. For example, the interfaces [Comparable](#) and [Cloneable](#) are implemented by many unrelated classes.
 - You want to specify the behavior of a particular data type, but not concerned

Object Oriented Analysis and Design using Java

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
- **Cannot be final** in Java because abstract classes are used only by *extending*
- A class **can extend only one abstract class** as Java does not support multiple inheritance

Coding Example: If the abstract class contains the below data, how to implement Rectangle and Triangle classes? (abstractdemo.java)

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

```
class Rectangle extends Figure {  
    ??  
}
```

```
class Triangle extends Figure {  
    ??  
}
```



THANK YOU

Prof. Mahitha G and Bhargavi M

Department of Computer Science and Engineering



Object Oriented Analysis and Design with Java - UE21CS352B

Prof Mahitha G and Bhargavi M
Ack:Prof. Priya Badarinath

Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

Overloading of Methods

Prof. Priya Badarinath

Department of Computer Science and Engineering

Object Oriented Analysis and Design with Java

Agenda



1. Introduction
2. Coding examples

Object Oriented Analysis and Design with Java

Method Overloading



Introduction

- A feature that allows a class to have more than one method having the same name, if the 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 with Java

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.

Object Oriented Analysis and Design with Java

Method Overloading



Coding Example 1:overloading.java

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



THANK YOU

Prof. Priya Badarinath

Department of Computer Science and Engineering

priyab@pes.edu





Object Oriented Analysis and Design using Java - UE21CS352

Prof.Mahitha G and Bhargavi M

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

Arrays and Collections(Arraylist and Stack)

Prof.Mahitha G & Bhargavi M

Department of Computer Science and Engineering

Object Oriented Analysis and Design using Java

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[];`

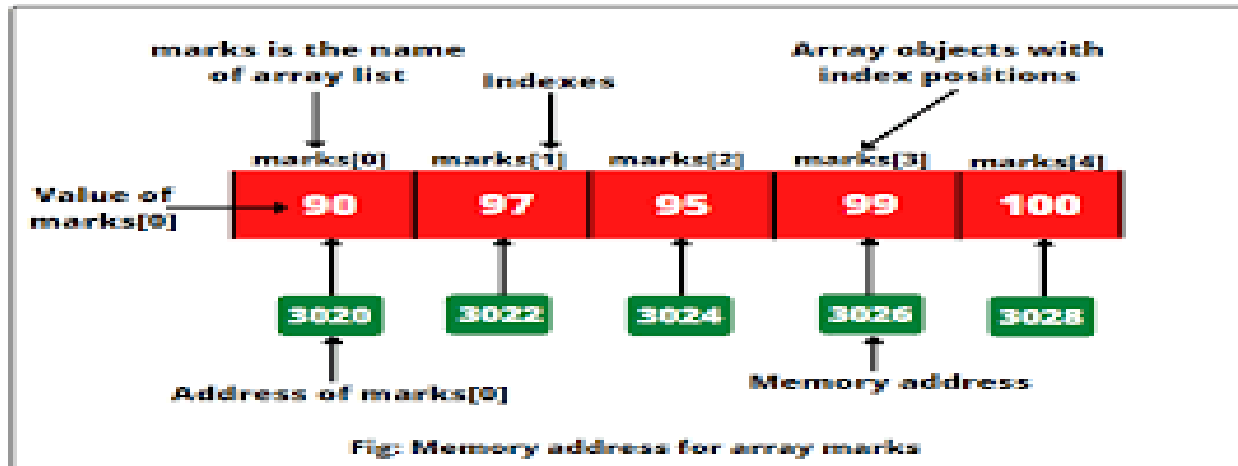
Instantiation:
`arrayRefVar=new datatype[size];`
- Declaration of 2D Array & Instantiation: :
 - `dataType[][] arr; (or)`
 - `dataType [][]arr; (or)`
 - `dataType arr [][];`

Instantiation:
`int[][] arr=new int[3][3]; //3 rows and 3 columns`

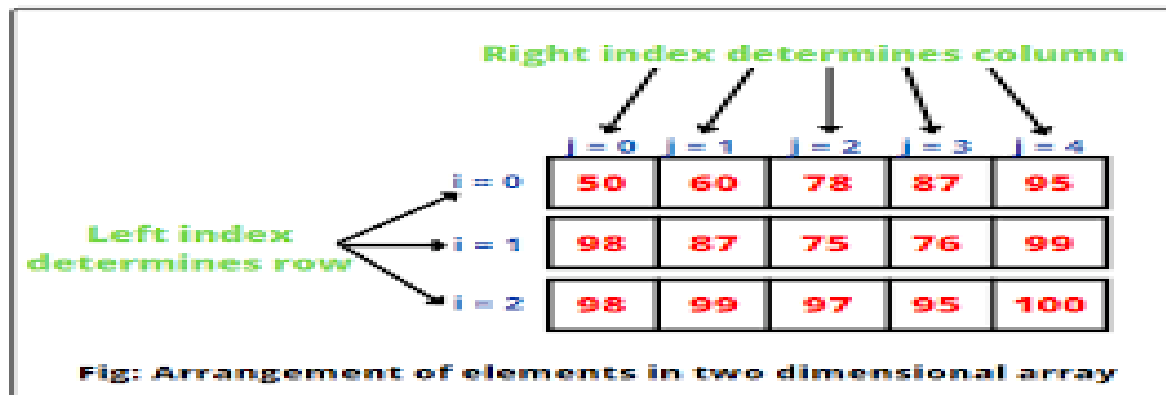
Object Oriented Analysis and Design using Java

Array Representation

Representation of 1 Dimensional Array



Representation of 2 Dimensional Array



Object Oriented Analysis and Design using Java

Arrays Class in Java

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

Usage :

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

Functions:

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

Object Oriented Analysis and Design using Java

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){ ...}.

Example program: foreach.java

Object Oriented Analysis and Design using Java

Collections - Introduction



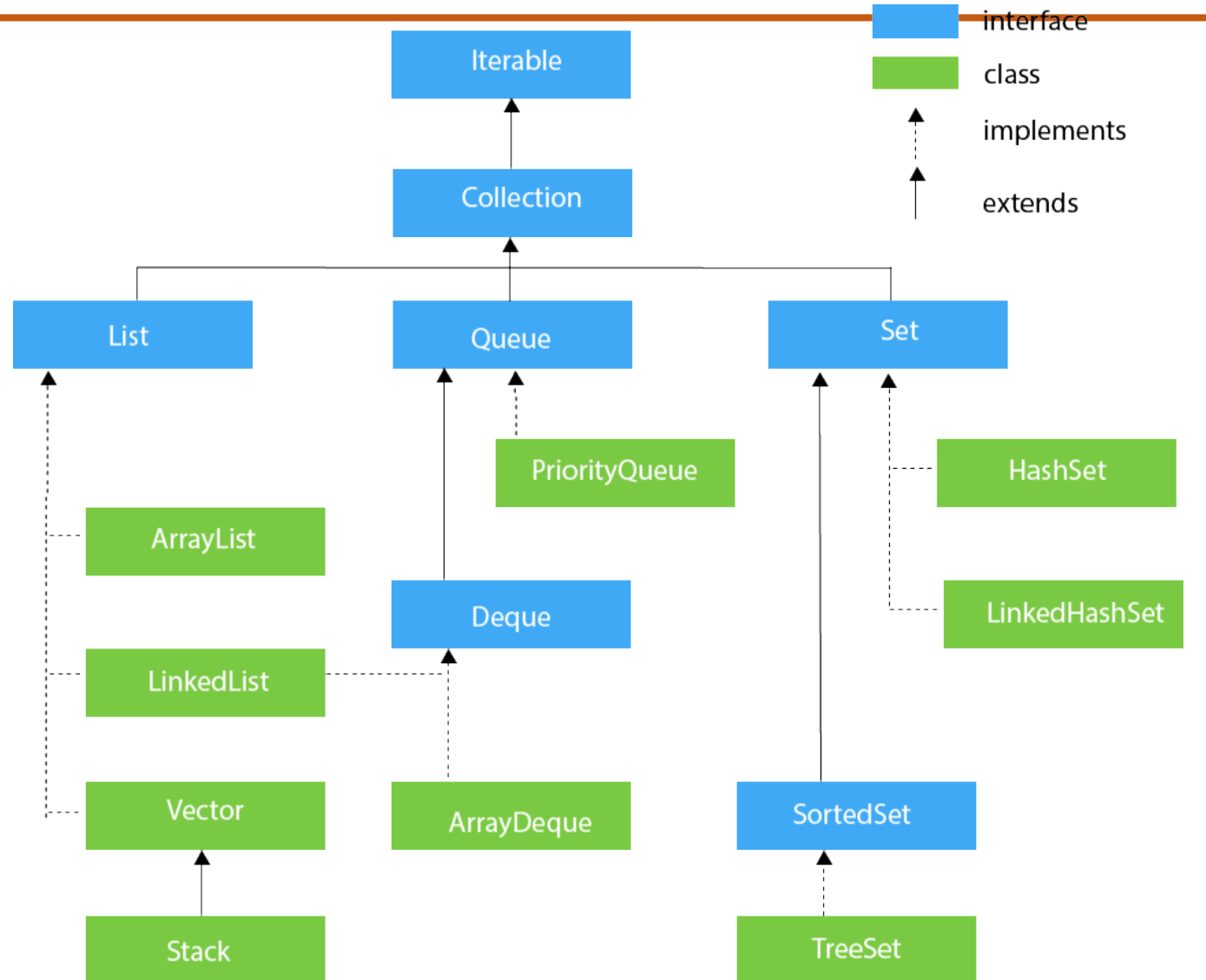
Collections framework. A *collection* is an object that represents a group of objects.

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 using Java

Collections - Introduction



Object Oriented Analysis and Design using Java

Collections - Introduction



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

```
public class Collections  
extends Object
```

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Methods in Collection Interface:

There are many methods declared in the Collection interface. Some of them are:

```
add  
addAll  
remove  
removeif  
size  
clear  
retainall
```

Object Oriented Analysis and Design using Java

List Interface



Interfaces List , Queue and Set are inherited from Collection class

List interface

List interface is the child interface of Collection interface.

List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.

To instantiate the List interface, we must use :

1. List <data-type> list1 = **new** ArrayList();
2. List <data-type> list2 = **new** LinkedList();
3. List <data-type> list3 = **new** Vector();
4. List <data-type> list4 = **new** Stack();

Object Oriented Analysis and Design using Java

Introduction to List Interface in Java

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

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

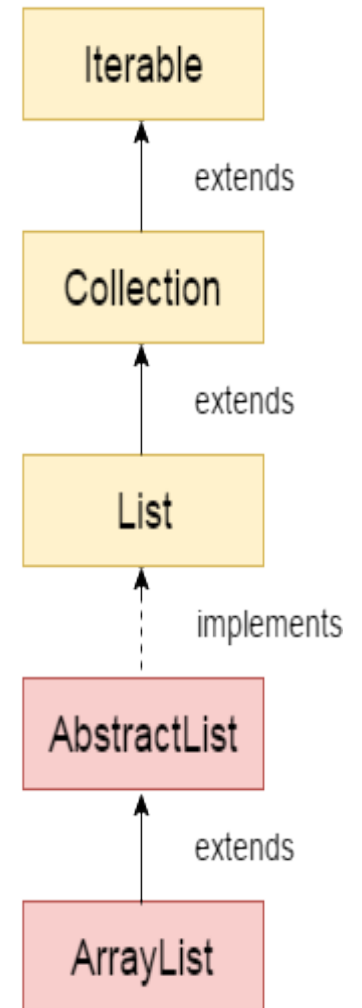
Few Methods :

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

Object Oriented Analysis and Design using Java

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 using Java

Introduction to ArrayList Class in Java

Declaration :

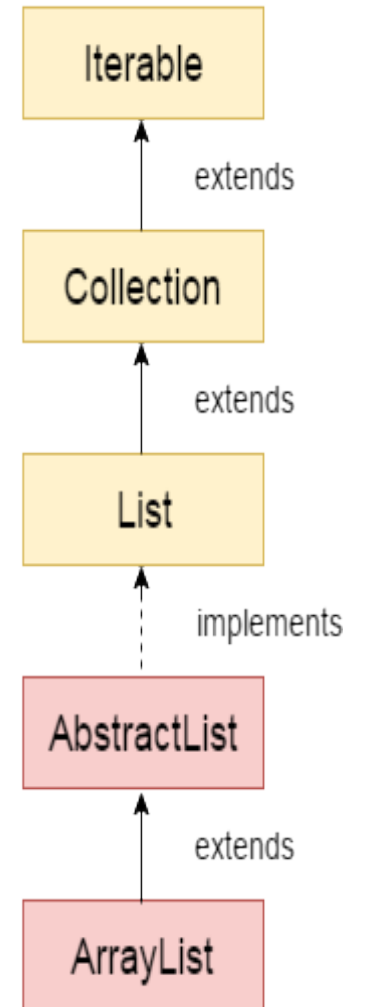
```
public class ArrayList<E> extends AbstractList<E> implements List<E>,  
RandomAccess, Cloneable, Serializable
```

Example for creating generic ArrayList:

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

Few Methods()

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



Object Oriented Analysis and Design using Java

Introduction to Stack Class in Java

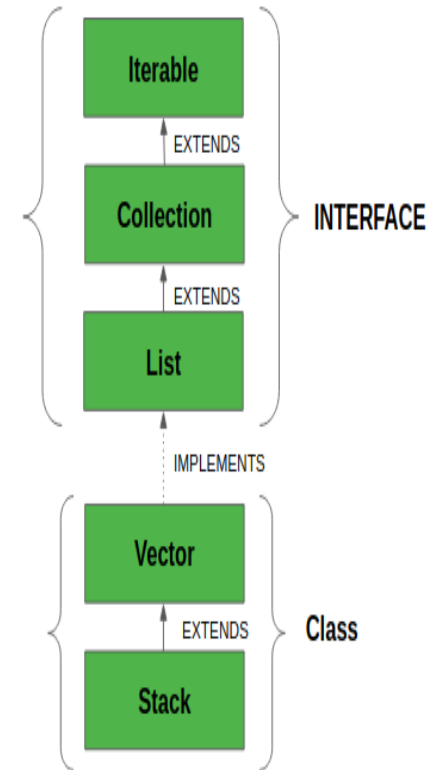
- Java Collection framework provides a Stack class that models and implements a **Stack data structure**.
- The class is based on the basic principle of last-in-first-out [LIFO].

Declaration:

```
public class Stack<E> extends Vector<E>
```

Implemented interfaces :

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



Object Oriented Analysis and Design using Java

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)`

Object Oriented Analysis and Design using Java

Programming Examples

- Programming Example:
ArrayList1.java
ArrayList2.java
Stack1.java



THANK YOU

Prof Mahitha G & Bhargavi M

Department of Computer Science and Engineering

mahithag@pes.edu