

CORE JAVA 8- DAY 4

Saratha Natarajan

AGENDA

- ◉ Today's Topic - OOPS, Wrapper Classes, Instance Of
- ◉ Exceptions

ENCAPSULATION

- ◉ a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.
- ◉ The **Java Bean** class is the example of a fully encapsulated class.

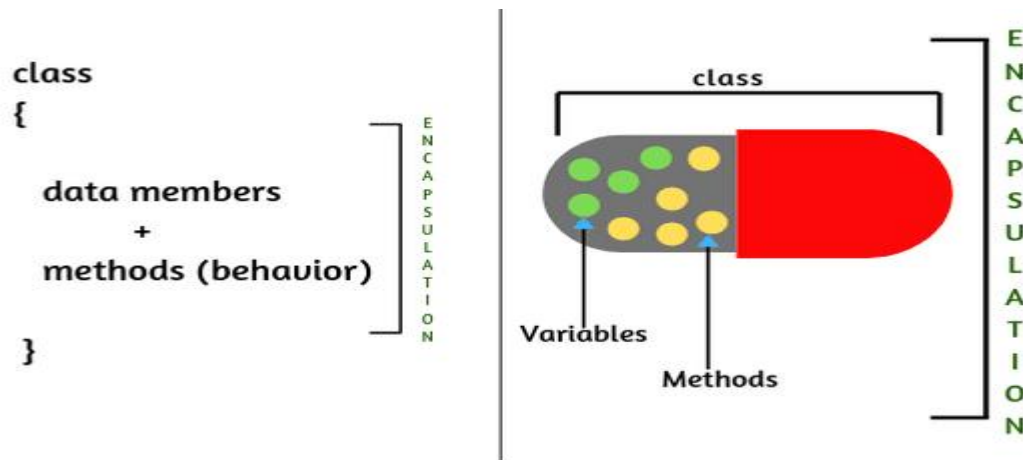


Fig: Encapsulation

ENCAPSULATION

◉ Realtime Example 1:

- School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc.

◉ Realtime Example 2:

- When you log into your email accounts such as Gmail, Yahoo Mail, or Rediff mail, there is a lot of internal processes taking place in the backend and you have no control over it.
- When you enter the password for logging, they are retrieved in an encrypted form and verified, and then you are given access to your account.
- You do not have control over it that how the password has been verified. Thus, it keeps our account safe from being misused.

INHERITANCE

- ◉ is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- ◉ idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- ◉ Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- ◉ Why use inheritance in java
 - For Method Overriding (so runtime polymorphism can be achieved).
 - For Code Reusability.

TERMS USED IN INHERITANCE

- ◉ **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- ◉ **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- ◉ **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- ◉ **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

DECLARE CLASSES(...)

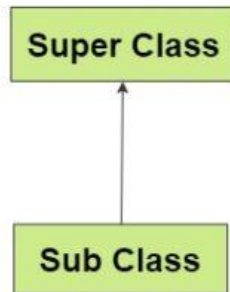
- ◉ The syntax of Java Inheritance
- ◉ **class** Subclass-name **extends** Superclass-name
 {
 //methods and fields
 }
- ◉ The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

INHERITANCE(...)

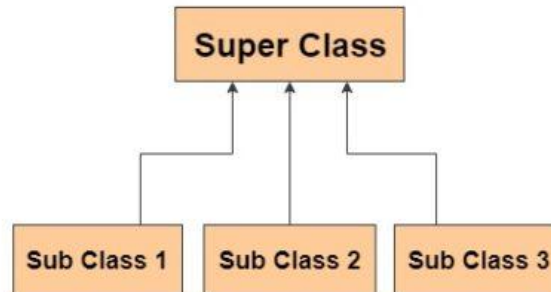
- Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

INHERITANCE - TYPES

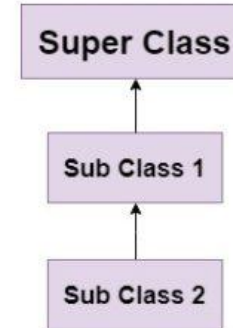
Single Inheritance



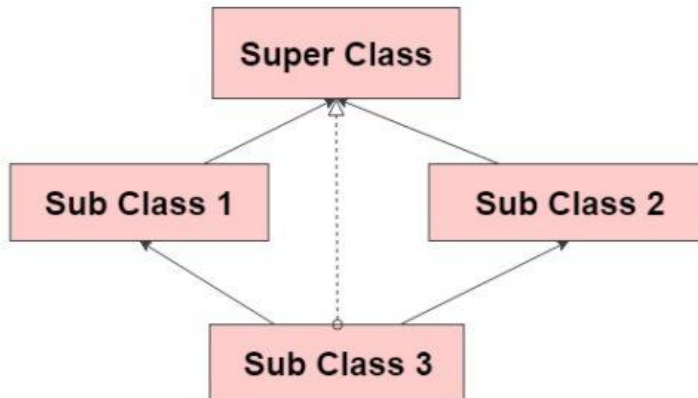
Hierarchial Inheritance



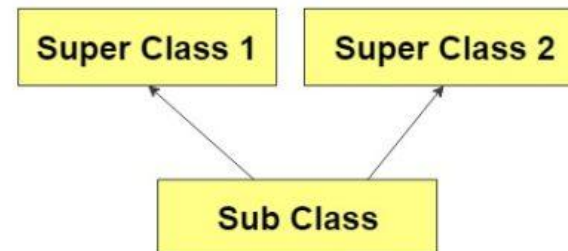
MultiLevel Inheritance

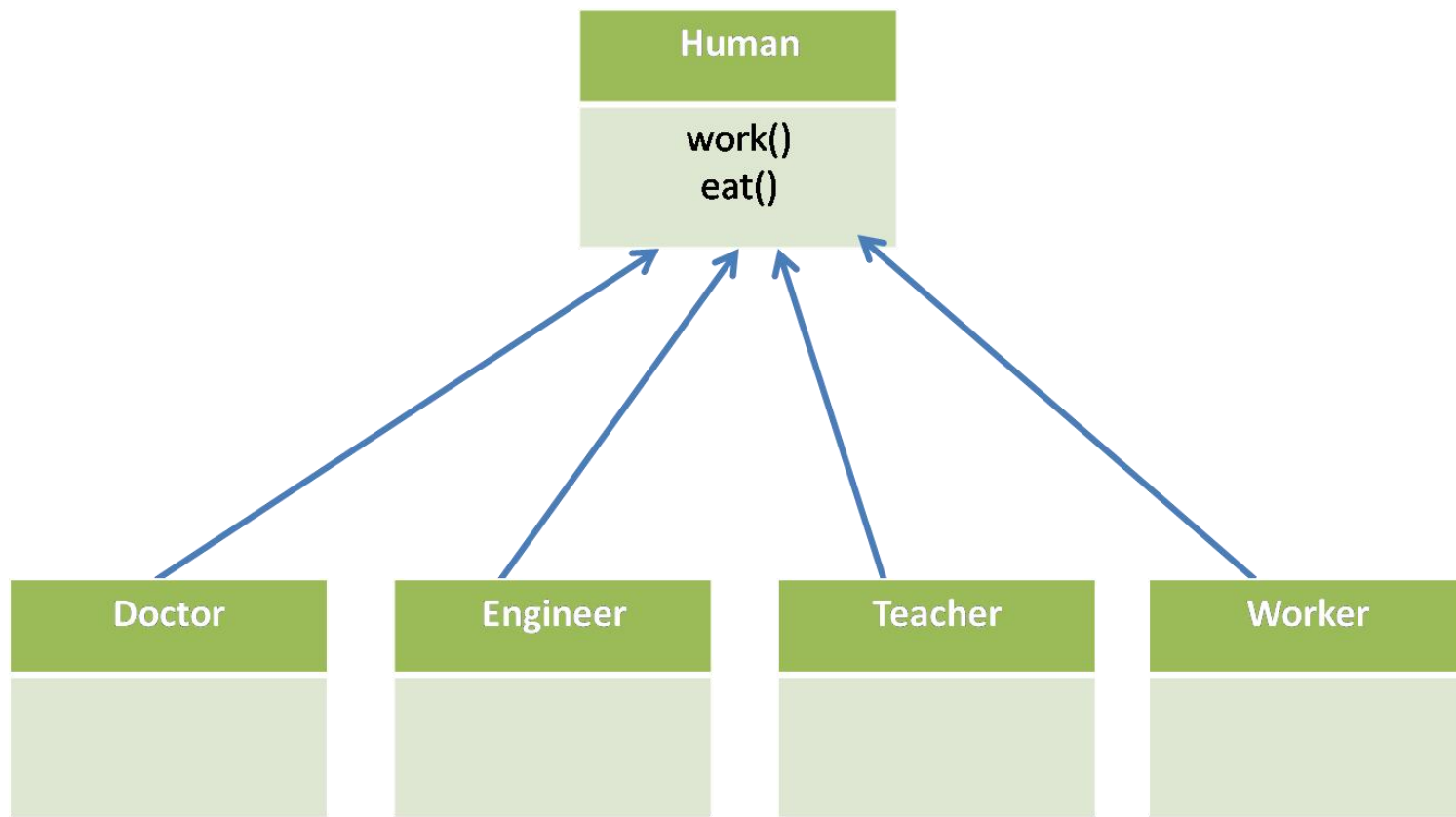


Hybrid Inheritance



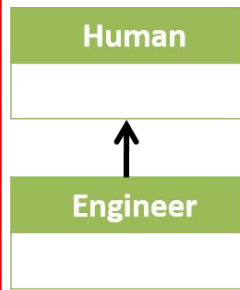
Multiple Inheritance



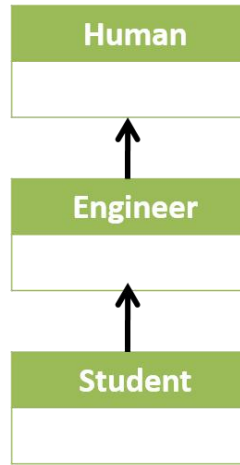


INHERITANCE(TYPES)

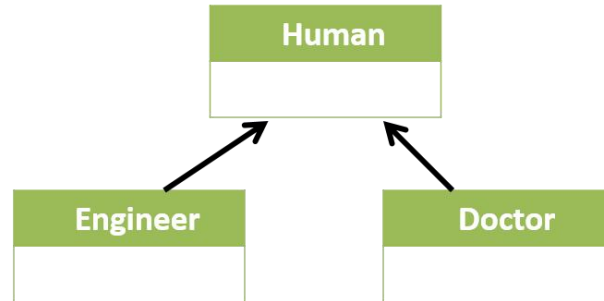
Single level Inheritance



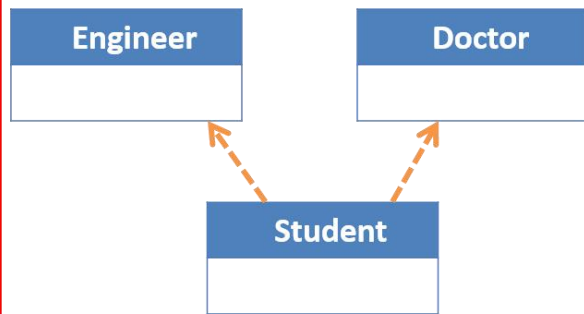
Multi level Inheritance



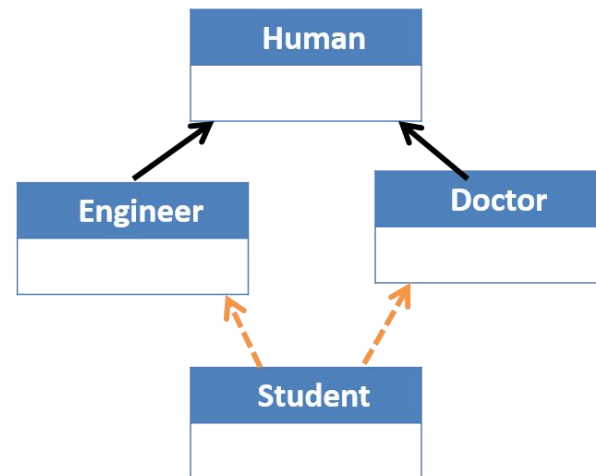
Hierarchical level Inheritance



Multiple level Inheritance



Hybrid level Inheritance



INHERITANCE(...)

⦿ “is-a” relationship

- means that a certain class is the part of the parent class.
- This is essential for code reusability. Inheritance is unidirectional which means that the child class is a type of parent class but the inverse is not true.
- For example, a brush is a tool but all tools are not brushes.
- This relationship gets implemented with the help of extends and implements keywords.

INHERITANCE(...)

⦿ “has-a” relationship

- Composition is the other word of the “has-a” relationship.
- It means that a certain instance of this class has a reference to another class or in the same class. This relationship also helps in code reusability.
- For example, A square has edges, A brush has bristles, A cat has a tail. This implementation is possible if we use the new keyword.

```
class Address
{
    .....
}
class Person
{
    Address addr = new Address();
    .....
    .....
}
```

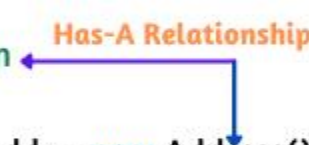
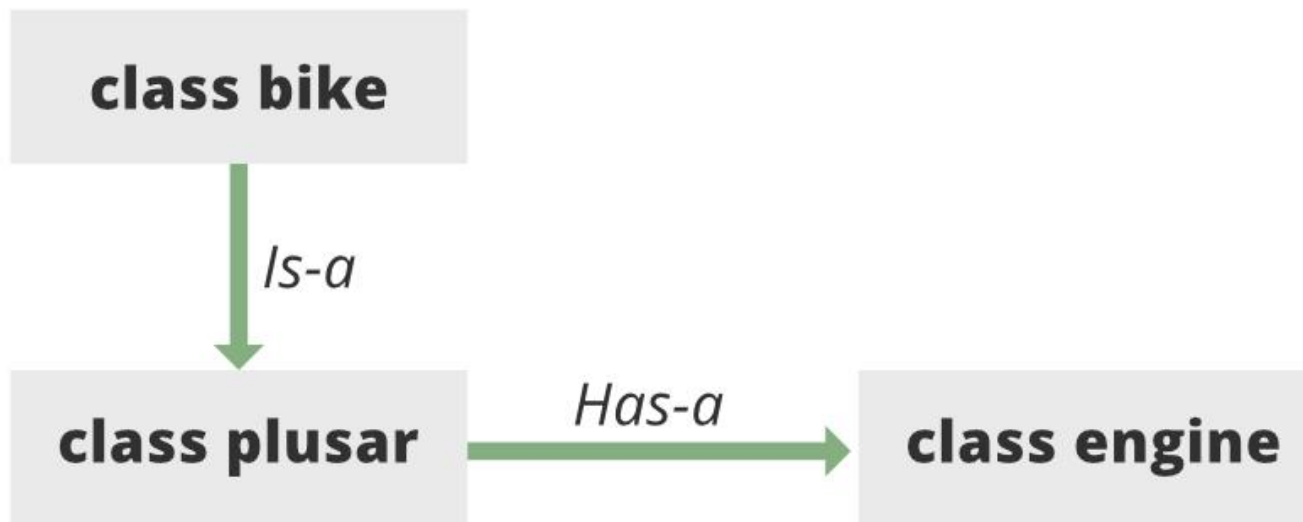
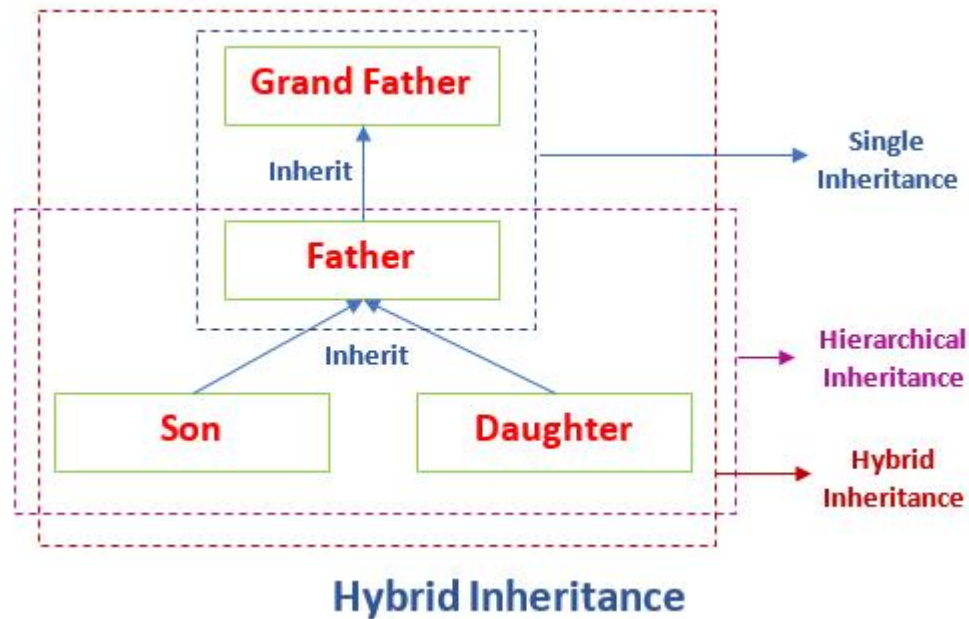


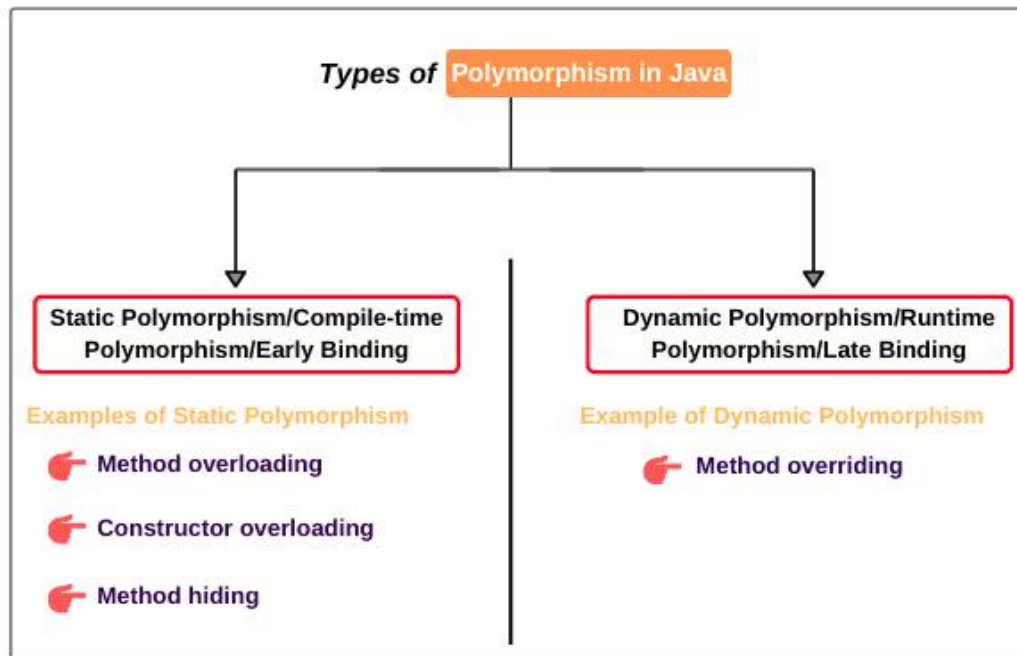
Fig: Has-A Relationship in Java



INHERITANCE(...)



POLYMORPHISM



- ◉ **Polymorphism in Java** is the task that performs a single action in different ways.
- ◉ You can perform Polymorphism in Java via two different methods:
 - Method Overloading
 - Method Overriding

What is Method Overloading in Java?

- is the process that can create multiple methods of the same name in the same class, and all the methods work in different ways. Method overloading occurs when there is more than one method of the same name in the class.

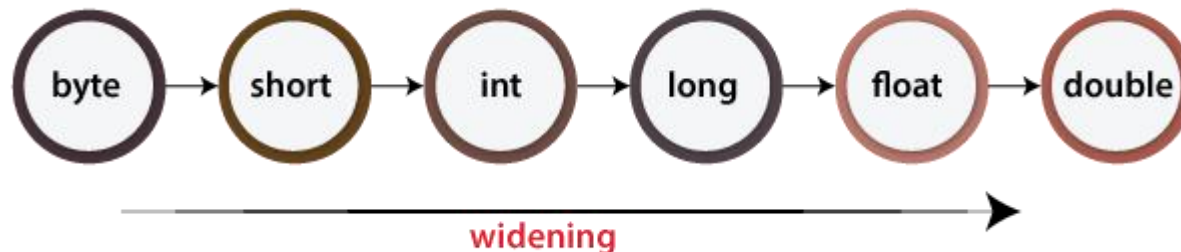
- ◉ Method overriding is the process when the subclass or a child class has the same method as declared in the parent class.

- ◉ Example

- Base class Polygon, -> Square , -> both with have render methods
- Base Class Language -> Java

IMPLICIT CONVERSION

- ◉ The process of converting one type of object and variable into another type is referred to as **Typecasting**. When the conversion automatically performs by the compiler without the programmer's interference, it is called **implicit type casting** or **widening casting**.



EXPLICIT CONVERSION

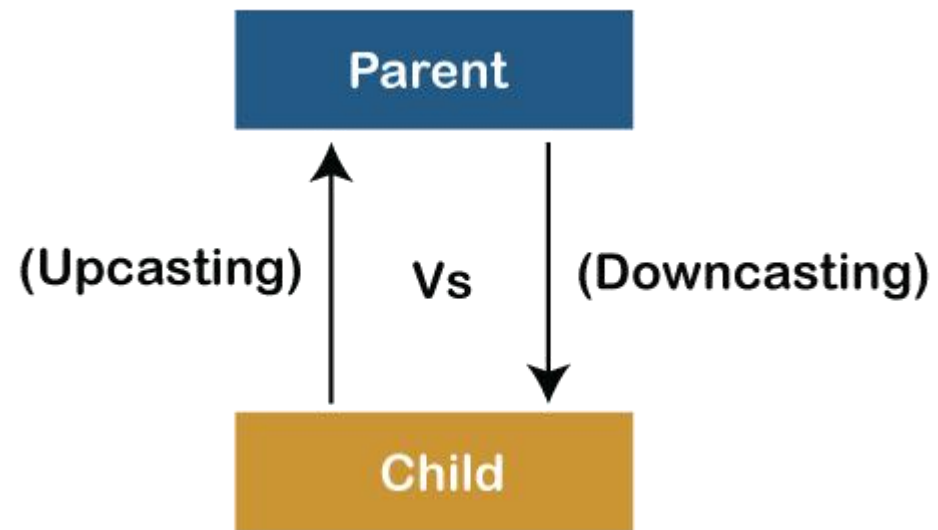
- ◉ If we want to assign a value of a larger data type to a smaller data type we perform explicit type casting or narrowing.
- ◉ This is useful for incompatible data types where automatic conversion cannot be done.
- ◉ Here, the target type specifies the desired type to convert the specified value to.

Double → Float → Long → Int → Short → Byte

Narrowing or Explicit Conversion

UPCASTING AND DOWNCASTING IN JAVA

- A process of converting one data type to another is known as **Typecasting** and **Upcasting** and **Downcasting** is the type of object typecasting.
- In Java, the object can also be typecasted like the datatypes.
- **Parent** and **Child** objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., **Child to Parent** and **Parent to Child** or can say **Upcasting** and **Downcasting**.



DOWNCASTING

- ◉ Downcasting
- ◉ **Downcasting** is another type of object typecasting. In **Downcasting**, we assign a parent class reference object to the child class.
- ◉ In Java, we cannot assign a parent class reference object to the child class, but if we per
- ◉ form downcasting, we will not get any compile-time error. However, when we run it, it throws the "**ClassCastException**". Now the point is if downcasting is not possible in Java, then why is it allowed by the compiler?
- ◉ In Java, some scenarios allow us to perform downcasting. Here, the subclass object is referred by the parent class.
- ◉ Below is an example of downcasting in which both the valid and the invalid scenarios are explained:

CLASS DECLARATIONS AND MODIFIERS (...)

◉ Non-Access Modifiers

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables.
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.

CLASS DECLARATIONS AND MODIFIERS (...)

◉ Static

- mainly used for memory management
- share the same variable or method of a given class
- apply static keywords with variables, methods, blocks, and nested classes.
- static keyword belongs to the class than an instance of the class.
- When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.

CLASS DECLARATIONS AND MODIFIERS (...)

◉ **Static blocks**

- can declare a static block that gets executed exactly once, when the class is first loaded.

◉ **Static variables**

- a single copy of the variable is created and shared among all objects at the class level
- Static variables are, essentially, global variables
- All instances of the class share the same static variable.

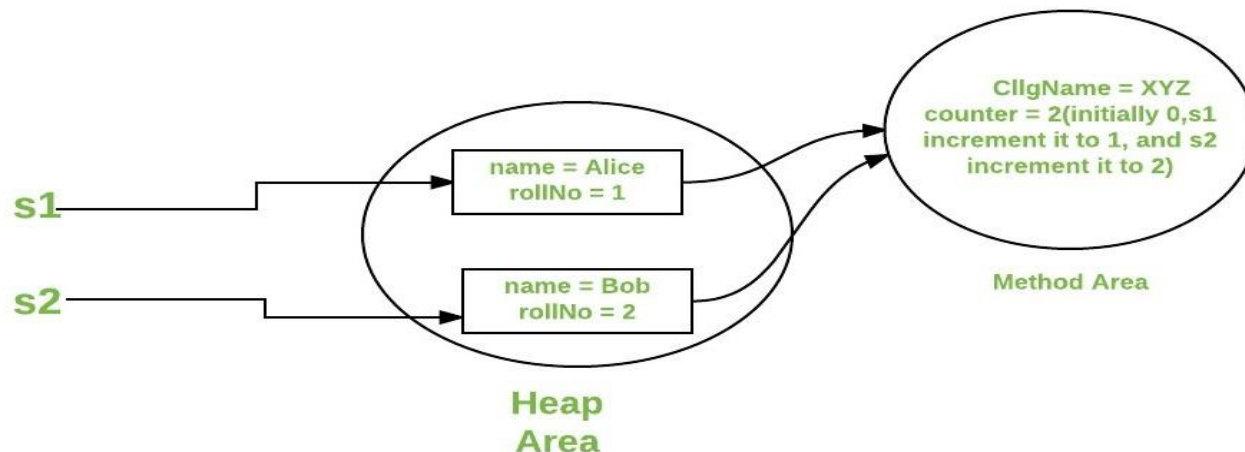
CLASS DECLARATIONS AND MODIFIERS (...)

◉ Static methods

- common example of a static method is the *main()* method
- Any static member can be accessed before any objects of its class are created, and without reference to any object
- Methods declared as static have several restrictions:
 - They can only directly call other static methods.
 - They can only directly access static data.
 - They cannot refer to this or super in any way.

CLASS DECLARATIONS AND MODIFIERS (...)

- ◉ **When to use static variables and methods?**
 - Use the static variable for the property that is common to all objects.
 - For example, in class Student, all students share the same college name.
 - Use static methods for changing static variables.



CLASS DECLARATIONS AND MODIFIERS (...)

◉ Final

- used to restrict the user.
- java final keyword can be used in many context. Final can be:
 - ◉ Variable
 - ◉ Method
 - ◉ class

◉ Java final variable - stop being modified

- you make any variable as final, you cannot change the value of final variable(It will be constant).
- Example
 - There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

- ◉ Final method - stop being overridden
 - If you make any method as final, you cannot override it.
- ◉ Java final class - stop being extended
 - If you make any class as final, you cannot extend it.

ABSTRACT

- ◉ *abstract* is a non-access modifier in java applicable for classes, methods but **not** variables. It is used to achieve abstraction which is one of the pillar of Object Oriented Programming(OOP).

ABSTRACT CLASSES

- ◉ The class which is having partial implementation(i.e. not all methods present in the class have method definition). To declare a class abstract, use this general form :
- ◉

```
abstract class class-name{  
    //body of class  
}
```
- ◉ Due to their partial implementation, we cannot instantiate abstract classes. Any subclass of an abstract class must either implement all of the abstract methods in the super-class, or be declared abstract itself.
- ◉ Some of the predefined classes in java are abstract. They depends on their sub-classes to provide complete implementation. For example, java.lang.Number is a abstract class.

ABSTRACT METHODS

- ◉ Sometimes, we require just method declaration in super-classes.
- ◉ This can be achieved by specifying the **abstract** type modifier.
- ◉ These methods are sometimes referred to as *subclasser responsibility* because they have no implementation specified in the super-class. Thus, a subclass must override them to provide method definition. To declare an abstract method, use this general form:
 - ◉ `abstract type method-name(parameter-list);`

INTERFACE

- ◉ is a blueprint of a class. It has static constants and abstract methods.
- ◉ you can say that interfaces can have abstract methods and variables. It cannot have a method body.
- ◉ Why use Java interface?
 - There are mainly three reasons to use interface. They are given below.
 - It is used to achieve abstraction.
 - By interface, we can support the functionality of multiple inheritance.
 - It can be used to achieve loose coupling.

- ◉ How to declare an interface?
- ◉ An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.
- ◉ Syntax:
- ◉ **interface** <interface_name>{
- ◉
- ◉ // declare constant fields
- ◉ // declare methods that abstract
- ◉ // by default.
- ◉ }
- ◉

DAY - 4 OVER

Activities- inheritance, static, abstract