INHERITANCE & POLYMORPHISM

OBJECTIVES

- Inheritance
- Polymorphism
- Final Classes
- Abstract Classes
- Interfaces

INHERITANCE

- A class can be derived from another class. The derived class is called as **sub class** and the class from where it is derived is called as **super class**
- Inheritance encourages **code reuse**
- Each subclass reuses the implementations in the super class and can add new responsibilities
- A subclass inherits all the members (fields, methods and nested classes) from its superclass.
- Constructors are not members, so they are not inherited.

WHY INHERITANCE?

```
class Student{

	String name;
	String state;

	String college;
	pursuedegree()
}
```

Redundant Code

```
class Employee{

	String name;
	String state;

	String companyName;
	promoteEmp()
}
```

INHERITANCE FOR REUSE

```
class Person{

      String name;
      String state;
}
```

```
class Student{

      String college;
      pursuedegree()
}
```

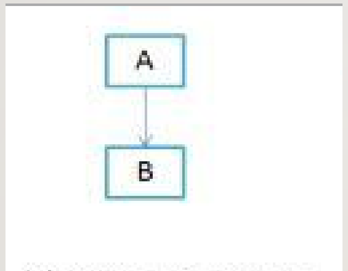
```
class Employee{

      String companyName;
      promoteEmp()
}
```

INHERITANCE (CONTD)

- Inheritance also means moving from generalization to specialization
- With access modifiers, we can restrict access of the super class members in subclass
- Excepting Object, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any explicit superclass, every class is implicitly a subclass of Object.

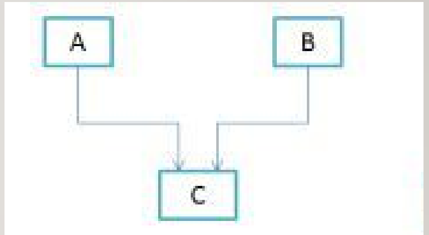
KIND OF INHERITANCE

- Single Inheritance



(a) Single Inheritance

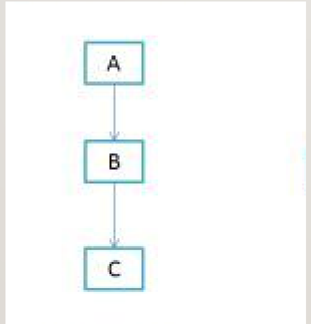
- Multiple Inheritance



(b) Multiple Inheritance

Java doesnot support Multiple Inheritance

- Multilevel Inheritance



(d) Multilevel Inheritance

HOW TO INHERIT IN JAVA

- Using extends keyword

```
class Person{}

class Student extends Person{}
```

- The Person class members can be used in Student class.

WHAT CAN BE DONE IN SUBCLASS

- A subclass inherits all of the public and protected members of its parent.
- You can use the inherited members as is, replace them, hide them, or supplement them with new members:
  - The inherited fields can be used directly, just like any other fields.
  - You can declare a field in the subclass with the same name as the one in the superclass, thus hiding it (not recommended).
  - You can declare new fields in the subclass that are not in the superclass.
  - The inherited methods can be used directly as they are.
  - You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it.
  - You can write a new static method in the subclass that has the same signature as the one in the superclass, thus hiding it.
  - You can declare new methods in the subclass that are not in the superclass.
  - You can write a subclass constructor that invokes the constructor of the superclass, either implicitly or by using the keyword super.

A subclass doesnot inherit the private members of its parent class.

CONSTRUCTOR CHAINING

- Every constructor calls its super class constructor directly using super() or indirectly using this()
- If the first statement of a constructor doesnot explicitly call this() or super(), the compiler adds the call to the default super constructor
  - If the superclass doesnot have a default constructor, compiler will throw an error.
- Every constructor in the complete inheritance hierarchy first gives a call to its super class constructor.
- Whenever any object is created, the very first constructor to fully get executed is of **java.lang.Object**

POLYMORPHISM

- One function Many implementations

Example:

1. Calculator doing addition operation which can take any type of input like int, double etc.

TYPES OF POLYMORPHISM

- Static Polymorphism
  - Method overloading
    - Same method name with different parameters
  - Method parameters define the type of method to be called
  - Same method

- Dynamic Polymorphism
  - Method overriding
    - Same prototype method defined in inherited classes

  - Type of Object determine the specific action

METHOD OVERLOADING

- Overloading means having multiple methods with the same name but different parameters
- The parameters must be different in any of the following ways
  - Number of parameters
  - Data types of parameters
  - Sequence of parameters
- Methods cannot be overloaded based on return type.

METHOD OVERRIDING

- Redefine the method in the subclass with the same method prototype defined in superclass.
- Used when the behavior of the child class is different from that of the base class
- Methods can not have less visibility than defined in the base class method, it can have more.
- Constructors cannot be overridden
- Static methods are not overridden, they are hidden.

Difference between hiding a static method and overriding an instance method has important implications:
 - The version of the overridden instance method that gets invoked is the one in the subclass
 - The version of the hidden static method that gets invoked depends on whether it is invoked from the superclass or the subclass

FINAL MODIFIER

- The final modifier is used with variables, methods and classes
- Final means constant, cannot be changed
- Final variable is constant
- Methods marked with final cannot be overridden
- Classes marked as final cannot be subclassed.

RUNTIME POLYMORPHISM – DYNAMIC METHOD DISPATCH

- Base class reference can hold derived class objects, it is called as upcasting

```
Employee emp = new Manager();
```

- The restriction with upcasting is that only members declared in the base class can be used, not the subclass members
- If we want to use the subclass members also, we need to do downcasting

```
Manager m = (Manager) emp;
```

Refer to: **ObjectTypeCast.doc**

DYNAMIC METHOD DISPATCH

- In case of overriding of methods, the definition of the subclass will be invoked. This is called as dynamic method dispatch.

- This is possible because of every method is being virtually invoked by the JVM.

ABSTRACT CLASS

- An abstract class is a class that is declared as 'abstract'.
- It cannot be instantiated but it can be subclassed
- The purpose of abstract class is to function as a base for subclass.
- An abstract class serves as a placeholder for common structure and behavior, which all its subclasses can use.
- Abstract class is declared with the keyword 'abstract'

```
abstract class Vehicle{}
```

- If we try to create an object of abstract class, it throws a compile time error

```
Vehicle v = new Vehicle();  // compile time error
```

ABSTRACT METHODS

- An abstract method is a method without implementation.

```
public abstract void testMethod();
```

- The class containing an abstract method must be declared as abstract but vice-versa is not true.
- An abstract class can contain both concrete methods as well as abstract methods
- Any class which subclasses Abstract class must implement all the abstract methods declared in the base class
- If the class doesnot want to override the abstract methods, the class must also be declared as abstract.
- Abstract methods serves as a contract between design and implementation
- During design phase, it is convenient to identify and design abstract methods and defer the implementation details to the development phase.

INTERFACES

- An interface is a reference type, similar to a class, that can contain only constants and method signatures
- Interfaces cannot be instantiated – they can only be implemented by classes or extended by other interfaces.
- An interface is defined with the help of keyword – ***interface***.
- All methods declared inside an interface are by default public, static and final
- A class implements an interface, making use of the keyword '***implements***'
- The class that implements an interface has to provide the definition of all the methods defined in the interface.

WHY INTERFACES?

- Interface as a type
    - When we define a new interface, we are define a new reference data type

- Interface as a contract
    - An interface serves as a contract between design and implementation.
    - At design time, we can easily identify the functionality needed to be achieved.
    - Based on varied implementation, different classes will implement the interface in a different ways.

- Interface as polymorphism
    - Multiple classes even if they are unrelated can implement the same interface and override the methods, thereby providing polymorphism in unrelated classes also.

INTERFACE AND ABSTRACTION

- Multiple teams in a project can work in parallel and one group need to use the functionality provided by other groups without worrying about how the code is written
- Interface helps in bringing such an abstraction
- The class creators implement the interface and the class users simply use the interface.

PROGRAMMING TO AN INTERFACE

- It is always a good programming practice to program to an interface and not to the implementation

- The advantage is, a change in implementation would not have the impact on the client code.

INTERFACE AND MULTIPLE INHERITANCE

- A class can implement multiple interfaces
- The class which implements multiple interfaces must provide the implementation for all the methods defined in all the interfaces that it implements
- Interface implementation is a behavior use, unlike class which is a structural use.
- This feature is many times seen as Java's way of substituting multiple inheritance

INHERITANCE AMONG INTERFACES

- An interface can extend from one of more interfaces.

- The need of inheritance among the interfaces can be because of the given possible scenarios
  - If suppose we have to define an interface with some functionalities and we come to know that some of the functionalities are already defined in another interface
    - Should we define the methods again in new interface as well?
  - We define an interface with 3 methods, the classes implement the interface and define the methods. Later on because of requirement changes, we are required to include 2 more methods to the interface
    - If we choose to change the interface to include the new 2 methods, what will happen to the classes already implementing the interface

DEFAULT AND STATIC METHODS IN INTERFACES

DEFAULT METHODS

- Add default implementation for methods in an interface
- Provided for backward compatibility.
- Adding new methods in interface without breaking its implementation.
- Classes can choose to override the default method and provide its own implementation

DEFAULT METHODS (CONTD)

- Syntax to declare default method:

```
default void someMethod(){
    //to do statements
}
```

- Two interfaces being implemented in same class, having same default method
  - The class must override the default method and give its own definition.
  - Still if needed to call specific interface method use the below statement within the overridden method

```
<interface_name>.super.<method_name>();
```

EXAMPLE OF USAGE OF DEFAULT METHOD

- foreach method has been added in Iterable interface

```
public interface Iterable<T> {
        public default void forEach(Consumer<? super T> consumer) {
          for (T t : this) {
              consumer.accept(t);
          }
        }
 }
```

→ foreach method is used in conjuction with Lambda expression.

POINTS ABOUT DEFAULT METHODS

Extend Interface without fear of breaking Implementation classes

Has bridge down differences between interfaces and abstract classes

Help us in avoiding utility classes, e.g Collections class

Help in removing base implementation classes

Cannot override a method from java.lang.Object class

Diamond Problem can occur if two base interfaces have the same default method.
Mandatory for implementing class to override default method

STATIC METHODS IN INTERFACE

- Definition in the interface only

- Used when one proper definition of any functionality is required.

EXAMPLE

```
interface Validation{
    static boolean checkUserName(String str) {
          boolean valid = Pattern.matches("[a-zA-Z]{5,8}", str);
          return valid;
    }
}
```

- This method can be used to validate UserName in any class by calling

```
            Validation.checkUserName("someName");
```

POINTS ABOUT STATIC METHODS

- Don't belong to a particular object. They are part of interface.
- Defining a static method in an interface is similar to defining a method in class.
- Can be called from another static or default methods.
- Can be used to provide utility methods like Collection sorting, validation check etc.
- Does not allow implementation classes to override them, thus providing security.
- Cannot be defined for Object class methods.
- Same prototype method in implementation class is considered as new method not the overridden method.