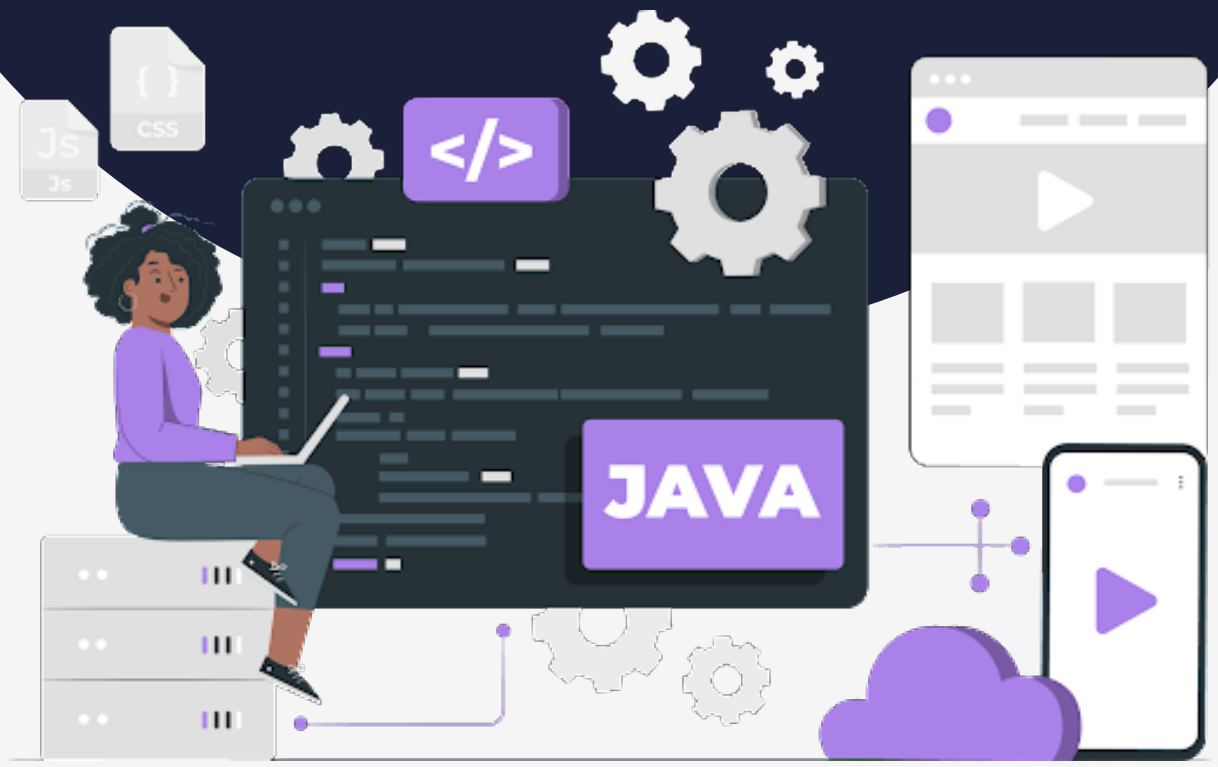
Lesson:

Polymorphism and Abstraction



List of Concepts Involved:

- What is polymorphism?
- How to achieve polymorphism
- Runtime vs Compile time polymorphism
- Abstract keyword and Abstraction
- Abstract class and Abstract method
- final class
- final variable
- final method

What is polymorphism?

If one thing exists in more than one form then it is called Polymorphism.
Polymorphism is a Greek word, where Poly means many and morphism means structures or forms.

1.Static Polymorphism
2.Dynamic Polymorphism

1.Static Polymorphism:
If polymorphism exists at compilation time then it is called Static Polymorphism.
Ex: Overloading.

2.Dynamic Polymorphism:
If the polymorphism exists at runtime then that polymorphism is called Dynamic Polymorphism.
Ex: Overriding

Method Overriding:

The process of replacing existing method functionality with some new functionality is called Method Overriding.

To perform Method Overriding, we must have inheritance relationship classes.
In Java applications, we will override super class method with sub class method.
In Java applications,we will override super class methods with subclass methods.
If we want to override super class method with subclass method then both super class method and sub class method must have the same method prototype.

Steps to perform Method Overriding:

1. Declare a superclass with a method which we want to override.
2. Declare a subclass and provide the same super class method with different implementation.
3. In the main class,in the main() method,prepare an object for the subclass and prepare a reference variable for super class[UpCasting].
4. Access the super class method then we will get output from the sub class method.

Example

```
class Loan{
	public float getIR(){
		return 7.0f;
	}
}
class GoldLoan extends Loan{
	public float getIR(){
		return 10.5f;
	}
}
class StudyLoan extends Loan{
	public float getIR(){
		return 12.0f;
	}
}
class CraftLoan extends Loan{
}
class Test{
	public static void main(String[] args){
		Loan gold_Loan=new GoldLoan();
		System.out.println("Gold Loan IR :"+gold_Loan.getIR()+"%");

		Loan study_Loan=new StudyLoan();
		System.out.println("Study Loan IR :"+study_Loan.getIR()+"%");

		Loan craft_Loan=new CraftLoan();
		System.out.println("Craft Loan IR :"+craft_Loan.getIR()+"%");
    }
}
```

NOTE:

To prove method overriding in Java, we have to access the super class method but JVM will execute the respective sub class method and JVM has to provide output from the respective sub classmethod, not from superclass method. To achieve the above requirement we must create reference variables for only super classes and we must create objects for subclasses.

```
class A{
	void m1(){
		System.out.println("m1-A");
	}
}
class B extends A{
	void m1(){
		System.out.println("m1-B");
	}
}
public class Test{
	public static void main(String args[]){
		A a  = new B();
		  a.m1();
	}
}
```

Rules to perform Method Overriding:

1.To override super class method with sub class, then super class method must not be declared as private.

Ex:

```
class A{
        private void m1(){
                System.out.println("m1-A");
        }
}
class B extends A{
      void m1(){
              System.out.println("m1-B");
      }
}

public class Test{
        public static void main(String args[]){
                    A a=new A();
                    a.m1();
        }
}
```

2.To override super class method with sub class method then sub class method should have the same return type of the super class method.

EX:

```
class A{
      int m1(){
              System.out.println("m1-A");
              return 10;
      }
}
class B extends A{
      void m1(){
              System.out.println("m1-B");
      }
}
public class Test{
       public static void main(String args[]){
               A a=new B();
               a.m1();
       }
}
```

3.To override super class method with sub class method then super class method must not be declared as final sub class method may or may not be final.

EX:

```
class A{
      void m1(){
              System.out.println("m1-A");
      }
}
class B extends A{
        final void m1(){
               System.out.println("m1-B");
        }
}
public class Test{
      public static void main(String[] args){
             A a=new B();
            a.m1();
      }
}
```

4.To override superclass method with sub class method either super class method or subclass method as static then compiler will rise an error.If we declare both super and sub class method as static in method overriding compiler will not rise any error,JVM will provide output from the super class Method.

NOTE: If we are trying to override superclass static method with subclass static method then super class static method will override subclass static method,where JVM will generate output from super class static method.

EX:

```
class A{
       static void m1(){
              System.out.println("m1-A");
       }
}
class B extends A{
       static void m1(){
              System.out.println("m1-B");
       }
}
public class Test{
       public static void main(String args[]){
                     A a=new B();
                     a.m1();
       }
}
```

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
class Parent{
        public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
          public void methodTwo(){System.out.println("methodTwo from child");}
}

public class TestApp{
       public static void main(String... args){
                Parent p=new Parent();
               p.methodOne();

                Child c=new Child();
               c.methodOne();//inherited method
                c.methodTwo();//Specialized method

                Parent p1=new Child();
                p1.methodOne();
                p1.methodTwo();//CE: can't find the symbol methodTwo in Parent
       }
    }
```

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

```
class Parent{
    public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
        @Override
        public void methodOne(){System.out.println("methodOne from child");}
 }
 public class TestApp{
        public static void main(String... args){
              Parent p=new Parent();
               p.methodOne();//methodOne from parent

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

Inherited method

- The method which would come from parent to child due to inheritance is called inherited method.

Example

```
class Parent{
        public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
          public void methodTwo(){System.out.println("methodTwo from child");}
}

public class TestApp{
       public static void main(String... args){
                Parent p=new Parent();
               p.methodOne();

                Child c=new Child();
               c.methodOne();//inherited method
                c.methodTwo();//Specialized method

                Parent p1=new Child();
                p1.methodOne();
                p1.methodTwo();//CE: can't find the symbol methodTwo in Parent
       }
    }
```

Overridden Method

The method which is taken from Parent and changes the implementation as per the needs of the requirement in the class is called the “overridden method”.

Example

```
class Parent{
    public void methodOne(){System.out.println("methodOne from parent");}
}
class Child extends Parent{
        @Override
        public void methodOne(){System.out.println("methodOne from child");}
 }
 public class TestApp{
        public static void main(String... args){
              Parent p=new Parent();
               p.methodOne();//methodOne from parent

             Child c=new Child();
              c.methodOne();//methodOne from child
        }
   }
```

5.To override super class method with subclass method,sub class method must have either the same scope of the super class method or more scope when compared with super class method scope otherwise the compiler will raise an error.

EX:

```
class A{
      protected void m1(){
         System.out.println("m1-A");
      }
}
class B extends A{
      public void m1(){
             System.out.println("m1-B");
      }
}
public class Test{
       public static void main(String args[]){
              A a=new A();
              a.m1();
       }
}
```

6.To override super class method with subclass method subclass method should have either same access privileges or weaker access privileges when compared with super class method access privileges.

CompileTime Polymorphism vs RunTime Polymorphism

What are the differences between method overloading and method overriding?

1. The process of extending the existing method functionality with new functionality is called Method Overloading.
   The process of replacing existing method functionality with new functionality is called Method Overriding.

2. In the case of method overloading, different method signatures must be provided to the methods
   In the case of method overriding, the same method prototypes must be provided to the methods.

3. With or without inheritance we can perform method overloading
   With inheritance only we can perform Method overriding

Abstract keyword and Abstraction

Example

```
class DB_Driver{
       public void getDriver(){
               System.out.println("Type-1 Driver");
       }
}
class New_DB_Driver extends DB_Driver{
      public void getDriver(){
             System.out.println("Type-4-Driver");
      }
}
class Test{
      public static void main(String args[]){
                    DB_Driver driver=new New_DB_Driver();
                    driver.getDriver();
     }
}
```

- In the above example, method overriding is implemented, in method overriding, for super class method call JVM has to execute subclass method, not super class method.
- In method overriding, always JVM is executing only subclass method, not super class method.
- In method overriding, it is not suggestible to manage super class method body without execution, so that, we have to remove superclass method body as part of code optimization.
- In Java applications, if we want to declare a method without body then we must declare that method as "Abstract Method".
- If we want to declare abstract methods then the respective class must be an abstract class.

Example

```
abstract class DB_Driver{
        public abstract void getDriver();
}
class New_DB_Driver extends DB_Driver{
      public void getDriver(){
              System.out.println("Type-4 Driver");
      }
}

public class Test{
       public static void main(String args[]){
                     DB_Driver driver=new New_DB_Driver();
                     driver.getDriver();
       }
}
```

- In Java applications, if we declare any abstract class with abstract methods, then it is convention to implement all the abstract methods by taking sub classes.
- To access the abstract class members, we have to create an object for the subclass and we have to create a reference variable either for abstract class or for the subclass.
- If we create reference variables for abstract class then we are able to access only abstract class members, we are unable to access subclass own members.
- If we declare a reference variable for subclass then we are able to access both abstract class members and subclass members.

Example

```
abstract class A{
        void m1(){
                System.out.println("m1-A");
        }

        abstract void m2();
        abstract void m3();
}
class B extends A{
      void m2(){
              System.out.println("m2-B");
      }
      void m3(){
              System.out.println("m3-B");
      }
      void m4(){
              System.out.println("m4-B");
      }
}

public class Test{
        public static void main(String args[]){
               A a=new B();
               a.m1();
               a.m2();
               a.m3();
               //a.m4();---error

               B b=new B();
               b.m1();
               b.m2();
               b.m3();
               b.m4();
       }
}
```

In Java applications,it is not possible to create Objects for abstract classes but it is possible to provide constructors in abstract classes,because,to recognize abstract class instance variables in order to store them in the subclass objects.

EX:

```
abstract class A{
  A(){
      System.out.println("A-Con");
  }
}
class B extends A{
      B(){
             System.out.println("B-Con");
      }
}
public class Test{
       public static void main(String[] args){
              B b=new B();
       }
}
```

In Java applications, if we declare any abstract class with abstract methods then it is mandatory to implement all the abstract methods in the respective subclass.
If we implement only some of the abstract methods in the respective subclass then compiler will rise an error, where to come out from compilation error we have to declare the respective subclass as an abstract class and we have to provide implementation for the remaining abstract methods by taking another subclass in multilevel inheritance.

EX:

```
abstract class A{
      abstract void m1();
      abstract void m2();
      abstract void m3();
}
abstract class B extends A{
       void m1(){
             System.out.println("m1-A");
       }
}
class C extends B{
      void m2(){
              System.out.println("m2-C");
     }
     void m3(){
              System.out.println("m3-C");
     }
}
public class Test{
       public static void main(String[] args){
              A a=new C();
              a.m1();
              a.m2();
              a.m3();
       }
}
```

In Java applications, if we want to declare an abstract class then it is not at all mandatory to have at least one abstract method, it is possible to declare abstract class without having abstract methods but if we want to declare a method as an abstract method then the respective class must be an abstract class.

```
abstract class A{
      void m1(){
              System.out.println("m1-A");
      }
}
class B extends A{
        void m2(){
                System.out.println("m2-B");
         }
}
public class Test{
       public static void main(String args[]){
              A a=new B();
              a.m1();
              //a.m2();---------->Error

               B b=new B();
               b.m1();
               b.m2();
      }
}
```

In Java applications, it is possible to extend an abstract class to concrete class and from concrete class to abstract class.

```
class A{
      void m1(){
             System.out.println("m1-A");
      }
}
abstract class B extends A{
        abstract void m2();
}
class C extends B{
        void m2(){
                System.out.println("m2-C");
        }
        void m3(){
                System.out.println("m3-C");
        }
}
public class Test{
       public static void main(String args[]){
              A a=new C();
              a.m1();
              //a.m2();  error
             //a.m3();  error
```

```
            B b=new C();
            b.m1();
            b.m2();
            //b.m3(); error

            C c=new C();
            c.m1();
            c.m2();
            c.m3();
    }
}
```

Note:
In Java applications, it is not possible to extend a class to the same class, if we do the same then the compiler will raise an error like "cyclic inheritance involving".

```
class A extends A{
}
Status: Compilation Error: “cyclic inheritance involving".

class A extends B{
}
class B extends A{
}
```

Status: Compilation Error: “cyclic inheritance involving".

final class

- If a class is marked as final, then the class won’t participate in inheritance,if we try to do so then it would result in “CompileTime Error”.

Eg: String, StringBuffer,Integer,Float,.....

final variable

- If a variable is marked as final, then those variables are treated as compile time constants and we should not change the value of those variables.
- If we try to change the value of those variables then it would result in “CompileTimeError”.

final method

- If a method is declared as final then those methods we can’t override, if we try to do so it would result in “CompileTimeError”.

Note
Every method present inside a final class is always final by default whether we are declaring or not. But every variable present inside a final class need not be final.

The main advantage of the final keyword is we can achieve security.
Whereas the main disadvantage is we are missing the key benefits of oops:
polymorsim (because of final methods), inheritance (because of final classes) hence if there is no specific requirement never recommended to use final keyboard.