2.6 Method Overriding

- ✓ When a method in a subclass has the same name and type signature as a method in its superclass, then the method in subclass is said to override a method in the superclass.

Example:

```
class Bank
    {
    int getRateOfInterest()// super class method
    {
    return 0;
    }
    }
    class Axis extends Bank// subclass of bank
    {
    int getRateOfInterest()// overriding the superclass method
    {
    return 6;
    }
    }
    class ICICI extends Bank// subclass of Bank
    {
            int getRateOfInterest()// overriding the superclass method
    {
    return 15;
    }
    }
    // Mainclass
    class BankTest
    {
    public static void main(String[] a)
    {
    Axis a=new Axis();
    ICICI i=new ICICI();
    // following method call invokes the overridden method of subclass AXIS
    System.out.println("AXIS: Rate of Interest = "+a.getRateOfInterest());
   // following method call invokes the overridden method of subclass ICICI
    System.out.println("ICICI: Rate of Interest = "+i.getRateOfInterest());
    }
    }
```

Output:

Z:\> java BankTest

```
AXIS: Rate of Interest = 6
ICICI: Rate of Interest = 15
```

- **RULES FOR METHOD OVERRIDING:**

  - ✓ The method signature must be same for all overridden methods.
  - ✓ Instance methods can be overridden only if they are inherited by the subclass.
  - ✓ A method declared final cannot be overridden.
  - ✓ A method declared static cannot be overridden but can be re-declared.
  - ✓ If a method cannot be inherited, then it cannot be overridden.
  - ✓ Constructors cannot be overridden.

- **ADVANTAGE OF JAVA METHOD OVERRIDING**

  - ✓ Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
  - ✓ Method Overriding is used for Runtime Polymorphism

2.7: DYNAMIC METHOD DISPATCH

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism.

Example that illustrate dynamic method dispatch:

```
class A {
void callme() {
System.out.println("Inside A's callme method");
}
}
class B extends A {
 //override callme()
void callme() {
System.out.println("Inside B's callme method");
} }

class C extends A
{
 //override callme()
void callme() {
System.out.println("Inside C's callme method");
}
}


class Dispatch
{
public static void main(String args[])
{
```

```
A a=new A();          //object of type A
B b=new B();          //object of type B
C c=new C();          //object of type C
A r;// obtain a reference of type A

r = a;      // r refers to an A object    // dynamic method dispatch
r.callme();// calls A's version of callme()

r = b;// r refers to an B object
r.callme();// calls B's version of callme()

r = c;// r refers to an C object
r.callme();// calls C's version of callme()
}
}
```

The output from the program is shown here:

```
Inside A's callme method
Inside B's callme method
Inside C's callme method
```

- **DIFFERENCE BETWEEN METHOD OVERLOADING AND METHOD OVERRIDING IN JAVA:**

	Method Overloading	Method Overriding
Definition	In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.	In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.
Meaning	Method Overloading means more than one method shares the same name in the class but having different signature.	Method Overriding means method of base class is re-defined in the derived class having same signature.
Behavior	Method Overloading is to "add" or "extend" more to method's behavior.	Method Overriding is to "Change" existing behavior of method.
Overloading and Overriding is a kind of polymorphism. Polymorphism means "one name, many forms".		
Polymorphism	It is a **compile time polymorphism**.	It is a **run time polymorphism**.
Inheritance	It may or may not need **inheritance** in Method Overloading.	It always requires inheritance in Method Overriding.
Signature	In Method Overloading, methods must have **different signature**.	In Method Overriding, methods must have **same signature**.
Relationship of Methods	In Method Overloading, relationship is there between methods of same class.	In Method Overriding, relationship is there between methods of super class and sub class.
No. of Classes	Method Overloading does not require more than one class for overloading.	Method Overriding requires at least two classes for overriding.

Example	Class Add
	{
	int sum(int a, int b)
	{
	return a + b;
	}
	int sum(int a)
	{
	return a + 10;
	}
	}
		Class A // Super Class
		{
		void display(int num)
		{
		print num ;
		}
		}
		//Class B inherits Class A
		Class B //Sub Class
		{
		void display(int num)
		{
		print num ;
		}
		}

2.8: ABSTRACT CLASSES

- **Abstraction:**

Abstraction is a process of hiding the implementation details and showing only the essential features to the user.

- ✓ For example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
- ✓ Abstraction lets you focus on what the object does instead of how it does it.

- **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

- **Abstract Classes:**

A class that is declared as abstract is known as **abstract class**. Abstract classes cannot be instantiated, but they can be subclassed.

✓ **Syntax to declare the abstract class:**

```
abstract class <class_name>
{
        Member variables;
        Concrete methods { }
        Abstract methods();
}
```

✓ Abstract classes are used to provide common method implementation to all the subclasses or to provide default implementation.

Properties of abstract class:

- abstract keyword is used to make a class abstract.
- Abstract class can't be instantiated.
- If a class has abstract methods, then the class also needs to be made abstract using abstract keyword, else it will not compile.
- Abstract classes can have both concrete methods and abstract methods.
- The subclass of abstract class must implement all the abstract methods unless the subclass is also an abstract class.
- A constructor of an abstract class can be defined and can be invoked by the subclasses.
- We can run abstract class like any other class if it has main() method.

Example:

```
abstract class GraphicObject {
   int x, y;
   ...
   void moveTo(int newX, int newY) {
      ...
   }
   abstract void draw();
   abstract void resize();
}
```

- **Abstract Methods:**

A method that is declared as abstract and does not have implementation is known as **abstract method.** It acts as placeholder methods that are implemented in the subclasses.

- ✓ **Syntax to declare a abstract method:**

```
abstract class classname
{

abstract return_type <method_name>(parameter_list);//no braces{}
        // no implementation required
                ……..
}
```

✓Abstract methods are used to provide a template for the classes that inherit the abstract methods.

Properties of abstract methods:

- The abstract keyword is also used to declare a method as abstract.
- An abstract method consists of a method signature, but no method body.
- If a class includes abstract methods, the class itself must be declared abstract.
- Abstract method would have no definition, and its signature is followed by a semicolon, not curly braces as follows:

```
public abstract class Employee {
  private String name;
  private String address;
  private int number;
  public abstract double computePay();
  //Remainder of class definition
}
```

- Any child class must either override the abstract method or declare itself abstract.

Write a Java program to create an abstract class named Shape that contains 2 integers and an empty method named PrintArea(). Provide 3 classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contain only the method PrintArea() that prints the area of the given shape.

```
abstract class shape
{
    int x, y;
    abstract void printArea();
}

class Rectangle extends shape
{
void printArea()
{
   System.out.println("Area of Rectangle is " + x * y);
```

```
}
}
class Triangle extends shape
{
void printArea()
{
System.out.println("Area of Triangle is " + (x * y) / 2);
}
}
class Circle extends shape
{
void printArea()
{
        System.out.println("Area of Circle is " + (22 * x * x) / 7);
}
}
class abs
{
public static void main(String[] args)
{
Rectangle r = new Rectangle();
r.x = 10;
r.y = 20;
r.printArea();

System.out.println(" ---------------------------------------------- ");

Triangle t = new Triangle();
t.x = 30;
t.y = 35;
t.printArea();

System.out.println(" ---------------------------------------------- ");
Circle c = new Circle();
c.x = 2;
c.printArea();

System.out.println(" ---------------------------------------------- ");
}
}
```

Output:

```
D:\>javac abs.java
D:\>java abs
Area of Rectangle is 200
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
Area of Triangle is 525
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
Area of Circle is 12
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -
```

2.9: final WITH INHERITANCE

What is final keyword in Java?

Final is a keyword or reserved word in java used for restricting some functionality. It can be applied to member variables, methods, class and local variables in Java.

- ✓ **final** keyword has three uses:
  1. For declaring variable – **to create a named constant.** A final variable cannot be changed once it is initialized.
  2. For declaring the methods – **to prevent method overriding**. A final method cannot be overridden by subclasses.
  3. For declaring the class – **to prevent a class from inheritance.** A final class cannot be inherited.

1. Final Variable:

Any variable either member variable or local variable (declared inside method or block) modified by final keyword is called final variable.

✓The final variables are equivalent to **const qualifier** in C++ and **#define** directive in C.

✓Syntax:

```
✓final data_type variable_name = value;
```

✓Example:

```
final int MAXMARKS=100;
final int PI=3.14;
```

✓The final variable can be assigned only once.

✓The value of the final variable will not be changed during the execution of the program. If an attempt is made to alter the final variable value, the java compiler will

throw an error message.

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.

```
1. class Bike
2. {
3.       final int speedlimit=90;//final variable
4.       void run( )
5.      {
6.         speedlimit=400;
7.      }
8.      public static void main(String args[])
9.     {
10.         Bike obj=new  Bike();
11.         obj.run();
12.    }
13.}
```

Output: Compile Time Error

NOTE: Final variables are by default read-only.

2. Final Methods:

- ✓ Final keyword in java can also be applied to methods.
- ✓ **A java method with final keyword is called final method and it cannot be overridden in sub-class.**
- ✓ If a method is defined with final keyword, it cannot be overridden in the subclass and its behaviour should remain constant in sub-classes.
- ✓ **Syntax:**

```
final return_type function_name(parameter_list)
{
// method body
}
```

- ✓
- ✓ **Example of final method in Java:**

```
1. class Bike
2. {
3.     final void run()
4.    {
5.         System.out.println("running");
6.    }
7.  }
8. class Honda extends Bike
9. {
10.  void run()
11. {
12.       System.out.println("running safely with 100kmph");
13. }
14. public static void main(String args[])
15. {
16.       Honda honda= new Honda();
17.       honda.run();
18. }
19.}
```

Output:

```
D:\>javac Honda.java
Honda.java:9: error: run() in Honda cannot override run() in Bike
            void run()
                 ^
       overridden method is final
       1 error
```

3. Final Classes:

- ✓ **Java class with final modifier is called final class in Java and they cannot be sub-classed or inherited.**
- ✓ **Syntax:**

```
final class class_name
{
    // body of the class
}
```

- ✓ Several classes in Java are final e.g. String, Integer and other wrapper classes.

✓ **Example of final class in java:**

```
1. final class Bike
2. {
3.  }
4. class Honda1 extends Bike
5. {
6.       void run()
7.     {
8.          System.out.println("running safely with 100kmph");
9.     }
10.    public static void main(String args[])
11.   {
12.         Honda1 honda= new Honda1();
13.         honda.run();
14.   }
15.   }
```

Output:

D:\>javac Honda.java
Honda.java:4: error: cannot inherit from final Bike class Honda extends Bike
^
1 error

Points to Remember:

1) A constructor cannot be declared as final.
2) Local final variable must be initializing during declaration.
3) All variables declared in an interface are by default final.
4) We cannot change the value of a final variable.
5) A final method cannot be overridden.
6) A final class cannot be inherited.
7) If method parameters are declared final then the value of these parameters cannot be changed.
8) It is a good practice to name final variable in all CAPS.
9) final, finally and finalize are three different terms. finally is used in exception handling and
10) finalize is a method that is called by JVM during garbage collection.