UNIT II

Defining a Class, Adding Variables and Methods, Creating Objects, Accessing Class Members, Constructors, methods Overloading, Static Members, Nesting of Methods. Inheritance: Extending a Class, Overriding Methods, Final Variables and Methods, Final Classes, Finalize Methods, Abstract methods and Classes, Visibility Control.

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Java Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction
Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation in Java OOPs Concepts
Encapsulation
Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Java Naming Convention

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

Fields
Methods
Constructors
Blocks
Nested class and interface

Syntax to declare a class:

```
class <class_name>
{
   field;
   method;
}
```

```
class Student
{
   Int id;

}
```

Instance variable in Java
A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

```
 //Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student
{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[])
{
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through
reference variable
  System.out.println(s1.name);
 }
}
```

Define class
Create object
Access member of class
Define methods in the class

```java
package org.example;
2 usages
public class Student {
    2 usages
    int id;String class_name;
}class  Stu
{    public static void main(String[] args) {
        Student s1=new Student();//creating an object of
         s1.id=101;s1.class_name="MSC(BT)";
        System.out.println(s1.id);//accessing member thro
```

Object and Class Example: Initialization through method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n)
{
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+"
"+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

```
class Employee{
   int id;
   String name;
   float salary;
   void insert(int i, String n, float s) {
      id=i;
      name=n;
      salary=s;
   }
   void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
   Employee e1=new Employee();
   Employee e2=new Employee();
   Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
      e2.insert(102,"irfan",25000);
      e3.insert(103,"nakul",55000);
      e1.display();
      e2.display();
      e3.display();
   }
   }
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
It is a special type of method which is used to initialize the object.
Every time an object is created using the new() keyword, at least one constructor is called.
It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

There are two rules defined for the constructor.

Constructor name must be the same as its class name
A Constructor must have no explicit return type
A Java constructor cannot be abstract, static, final, and synchronized

Rules for creating Java constructor

There are two rules defined for the constructor.

Constructor name must be the same as its class name
A Constructor must have no explicit return type
A Java constructor cannot be abstract, static, final, and synchronized

Lab on Java Programming:

1. Working with Objects, Arrays, Conditionals and Loops.
2. Creating Classes and Applications in Java.
3. Java Exception handling
4. Streams and I/O, Using Native Methods and Libraries
5. Simple Animation and Threads, Advanced Animation, Images and Sound.
6. Managing Simple Events and Interactivity.
7. Local and global alignment of sequences
8. Creating User Interfaces with AWT, Modifiers.
9. Multithreading example
10. Java Programming Tools, Working with Data Structures.

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

//Java Program to create and call a default constructor

```
class Bike1{
//creating a default constructor
Bike1()
{System.out.println("Bike is created");
}
//main method
public static void main(String args[])

{
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

//Let us see another example of default constructor

```
//which displays the default values
class Student3
{
int id;
String name;
//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

//Java Program to demonstrate the use of the parameterized constructor.

```
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Java Copy Constructor

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

By constructor
By assigning the values of one object into another
By clone() method of Object class

In this example, we are going to copy the values of one object into another using Java constructor.

```
//Java program to initialize the values from one object to another object.
class Student6{
    int id;
    String name;
    //constructor to initialize integer and string
    Student6(int i,String n){
    id = i;
    name = n;
    }
    //constructor to initialize another object
    Student6(Student6 s){
    id = s.id;
    name =s.name;
    }
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){
    Student6 s1 = new Student6(111,"Karan");
    Student6 s2 = new Student6(s1);
    s1.display();
    s2.display();
   }
}
```

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

Variable (also known as a class variable)
Method (also known as a class method)
Block
Nested class

1) Java static variable
If you declare any variable as static, it is known as a static variable.

The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
The static variable gets memory only once in the class area at the time of class loading.
Advantages of static variable
It makes your program memory efficient (i.e., it saves memory).

Understanding the problem without static variable

```
class Student{
    int rollno;
    String name;
    String college="ITS";
}
```

Suppose there are 500 students in my college, now all instance data members will get memory each time when the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Example of static variable

```
//Java Program to demonstrate the use of static variable
class Student{
   int rollno;//instance variable
   String name;
   static String college ="ITS";//static variable
   //constructor
   Student(int r, String n){
   rollno = r;
   name = n;
   }
   //method to display the values
   void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();  }}
```

Program of the counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter{
int count=0;//will get memory each time when the instance is created

Counter(){
count++;//incrementing value
System.out.println(count);
}

public static void main(String args[]){
//Creating objects
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Program of counter by static variable
As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2{
static int count=0;//will get memory only once and retain its value

Counter2(){
count++;//incrementing the value of static variable
System.out.println(count);
}

public static void main(String args[]){
//creating objects
Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();
}
}
```

2) Java static method

If you apply static keyword with any method, it is known as static method.

A static method belongs to the class rather than the object of a class.

A static method can be invoked without the need for creating an instance of a class.

A static method can access static data member and can change the value of it.

Example of static method

```
/Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
    static void change(){
    college = "BBDIT";
    }
    //constructor to initialize the variable
    Student(int r, String n){
    rollno = r;
    name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+"
"+college);}
}
```

```
//Test class to create and display the values of object
public class TestStaticMethod{
    public static void main(String args[]){
    Student.change();//calling change method
    //creating objects
    Student s1 = new Student(111,"Karan");
    Student s2 = new Student(222,"Aryan");
    Student s3 = new Student(333,"Sonoo");
    //calling display method
    s1.display();
    s2.display();
    s3.display();
    }
}
```

Restrictions for the static method
There are two main restrictions for the static method. They are:

The static method can not use non static data member or call non-static method directly.
this and super cannot be used in static context.

This keyword in Java
There can be a lot of usage of Java this keyword. In Java, this is
a reference variable that refers to the current object.

Usage of Java this keyword
Here is given the 6 usage of java this keyword.

this can be used to refer current class instance variable.
this can be used to invoke current class method (implicitly)
this() can be used to invoke current class constructor.
this can be passed as an argument in the method call.
this can be passed as argument in the constructor call.
this can be used to return the current class instance from the method.

Inheritance:

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

For Method Overriding (so runtime polymorphism can be achieved).
For Code Reusability.

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.

The syntax of Java Inheritance

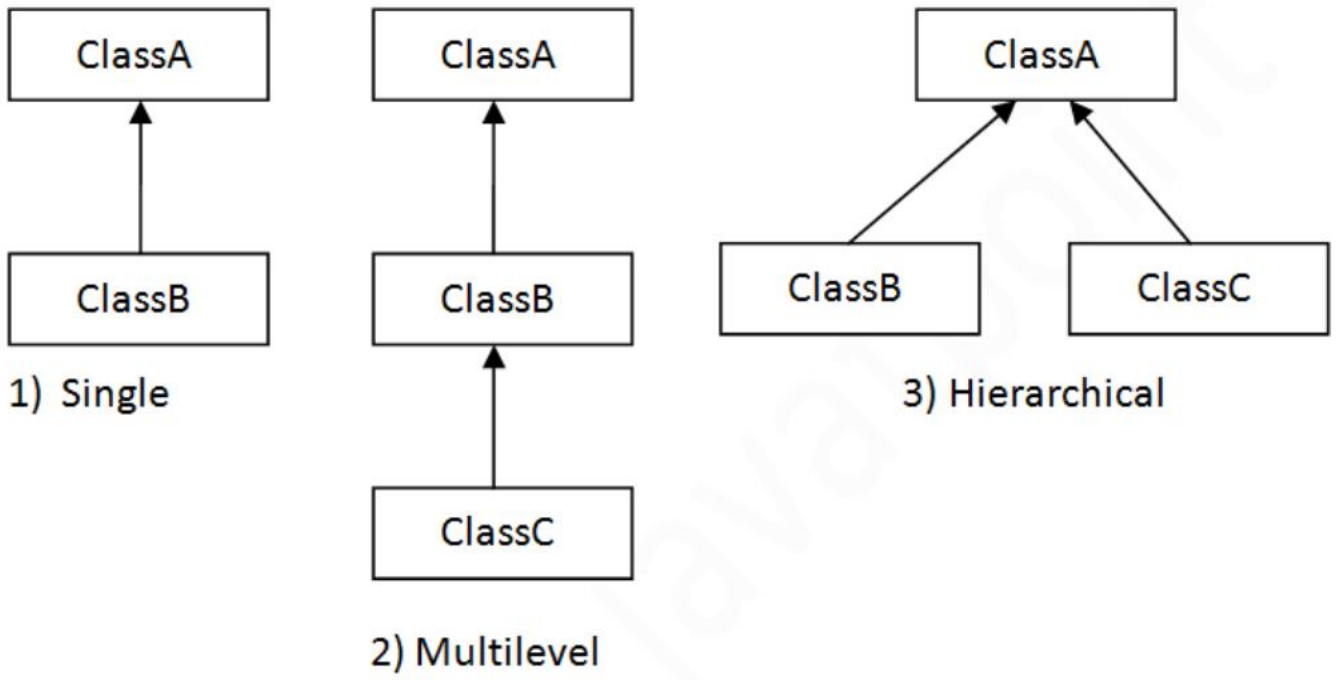
```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

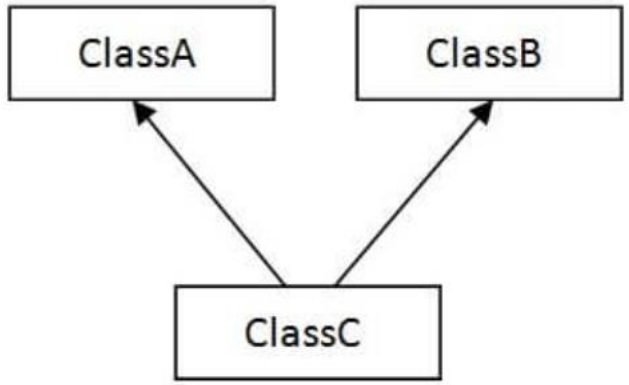
The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.

```
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);
}
}
```

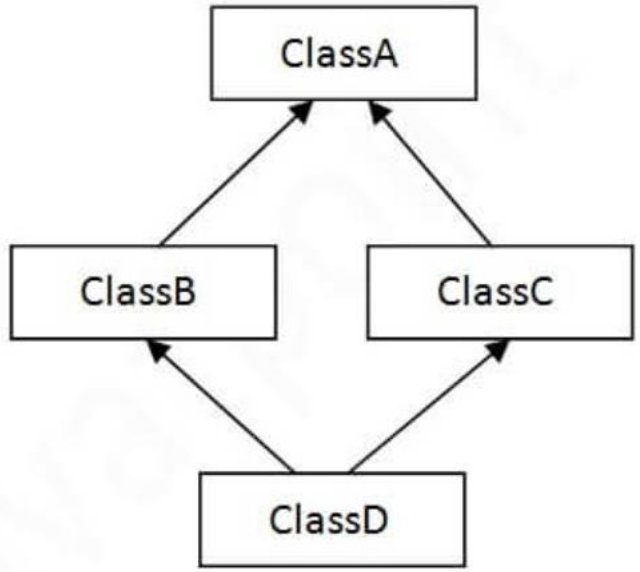
Types of inheritance in java
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.





4) Multiple



5) Hybrid

Single Inheritance Example

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Final Class in Java

As the name suggests, Final Class in Java uses the final keyword for its declaration. The final keyword in java is used to restrict the user, similarly, the final class means that the class cannot be extended. We can only create a final class if it is complete in nature, which means **it cannot be an abstract class**. All wrapper classes in Java are final classes, such as String, Integer, etc. Final class cannot be inherited by any subclass,

```
2 usages
public class Final_class {
    1 usage
    void myFinalMethod()
    {
        System.out.println("We are in the final class we just created")
    }
}
class MainClass
{
    public static void main(String arg[])
    {
        Final_class fc = new Final_class();
        fc.myFinalMethod();
    }
}
```

Example: 1 What happens if we try to inherit from a final Class?
In the below code, we are using the extend keyword to inherit the final class to the subClass but the final class cannot be inherited by any other subclass so it will throw an error saying "cannot inherit from final myFinalClass"

```
final class myFinalClass
{
   void myMethod()
   {
      System.out.println("We are in the final class we just created");
   }
}

class subClass extends myFinalClass
{
   void myMethod()
   {
      System.out.println("We are in the subclass");
   }
}
class MainClass
{
   public static void main(String arg[])
   {
      myFinalClass fc = new subClass();
      fc.myMethod();
   }
}
```

Advantage of the Final Class

Final class provides security as it cannot be extended or inherited by any other class, classes that can be extended can leak sensitive data of potential users but final classes make data elements safe and secure.

Final class is a complete and immutable class, so data elements do not change by external access.

Abstraction

Abstraction in Java is a process of hiding the implementation details from the user and showing only the functionality to the user. It can be achieved by using abstract classes, methods, and interfaces. An abstract class is a class that cannot be instantiated on its own and is meant to be inherited by concrete classes. An abstract method is a method declared without an implementation. Interfaces, on the other hand, are collections of abstract methods.

Introduction

Abstraction is a key concept in OOP and in general as well. Think about real world objects, they are made by combining raw materials like electrons, protons, and atoms, which we don't see due to the abstraction that nature exposes to make the objects more understandable. Similarly, in computer science, abstraction is used to hide the complexities of hardware and machine code from the programmer.

This is achieved by using higher-level programming languages like Java, which are easier to use than low-level languages like assembly.

Abstraction in Java

Abstraction in Java refers to hiding the implementation details of a code and exposing only the necessary information to the user. It provides the ability to simplify complex systems by ignoring irrelevant details and reducing complexity. Java provides many in-built abstractions and few tools to create our own.

Abstraction in Java can be achieved using the following tools it provides :

Abstract classes
Interfaces
Let's model the characters and objects from the very famous Angry Birds game to understand these concepts in detail.

Example
Basically, the gameplay involves shooting birds into pigs and structures in order to defeat them and retrieve their stolen eggs.

```
class Bird {
  String name;
  int size;
  int strength;
  protected void attack() {
    // empty method to be overridden by specific bird types
  }
  public String getName(){
    return name;
  }
}
class Hal extends Bird {
  @Override
  public void attack() {
    // implementation of boomerang attack for Hal object
  }
}class Chuck extends Bird {
  @Override
  public void attack() {
    // implementation of speed-up attack for Chuck object
  }
}// instantiate Chuck and Hal objects
Bird chuck = new Chuck();
Bird hal = new Hal();
// call attack method on objects
chuck.attack(); // Chuck speeds up
hal.attack(); // hal spins and boomerangs
```

The code models the characters and objects in the game Angry Birds. It defines a Bird class with properties such as name, size, and strength, and a method attack() to be overridden by specific bird types. The classes Hal and Chuck extend the Bird class and provide the implementation for the attack() method. The reference type for the Chuck and Hal objects is Bird, but polymorphism ensures that the right attack() method is called at runtime.

However, there are some problems with the above model. What if I do the following :

```
Bird someBird = new Bird();
someBird.attack();
```

Which character does this instantiated object define? The answer is none.

The Bird object does not define a character. It can be attacked but the attack() method will not do anything because the implementation of the attack is in the sub-classes. Bird is just a model of real birds and cannot be instantiated. To stop creating Bird objects and make the model clearer, it can be made into an abstract class using the abstract keyword.

What is an Abstract Class?

In Java, an abstract class is a class that cannot be instantiated on its own and is meant to be subclassed. It provides a common base for subclasses to inherit method and field definitions, but allows for subclasses to provide their own implementations.

We can mark the Bird class abstract and the attack method as abstract to understand it better:

```
class abstract Bird {

  String name;
  int size;
  int strength;
     // other properties go here

  protected abstract void attack();
     public String getName(){
       return name;
     }
}
```

Now if we create object of Bird class:

Bird bird = new Bird();

The compiler would complain and the code would not compile.

What is an Abstract Method?

An abstract method is a method that is declared but not defined in a class. It acts as a placeholder for methods that must be implemented in subclasses.

It is used to enforce a certain level of consistency across related classes and ensure that subclasses implement specific behaviors. By declaring a method as abstract, you specify that the method must be implemented in a subclass, but you do not provide an implementation for the method in the superclass.

```
protected abstract void attack();
```

Interface

Interfaces are probably the most powerful tools to achieve abstraction in java.

Do not confuse it with the GUI interface or the dictionary or generic use of the work interface meaning API. It's a java keyword called the interface.

Let's see how interfaces solve the above-mentioned problems.

Interfaces are completely pure abstract classes. As a result of this, the classes implementing an interface are forced to give the implementation of all the methods and hence the JVM is not confused about which implementation to pick at runtime.

Since it's just like an abstract class, it cannot be instantiated. And since it actually doesn't provide an implementation and it cannot be instantiated it can't have a constructor either. So let's define the Hittable interface for the GameObjects that can be hit:

```
public interface Hittable {
  int calculateDamage(int strength, double velocity, double angle);
}
```

Why do classes have to implement interfaces?

Duh, since all methods are abstract they don't have any body. So by themselves they just define an API or method signatures however some class has to supply its implementation for the defined methods or API.

Which classes should implement the interface?

In our example any game object that is-a Hittable entity or class can implement this interface. Meaning, that any class that can take up the role that the interface is defining can implement it. To implement an interface use the keyword implementsfollowed by the interface name like so :

```
public abstract class Bird extends GraphicObject implements Hittable {
  int calculateDamage(int strength, double velocity, double angle){
    // provide the logic to calculate damage for a bird
      }
}
```

Similarly the Block class can also implement the same interface
:

```
public abstract class Block extends GraphicObject implements Hittable {
  int calculateDamage(int strength, double velocity, double angle) {
    // provide the logic to calculate damage for a bird
      }
}
```

The compiler forces you to provide a body for the calculateDamage method as soon as the class implements the Hittable interface.

Let’s look at the Hittable interface definition a bit more closely.

```
public interface Hittable {
  int calculateDamage(int strength, double velocity, double angle);
}
```