Abstraction – It is the property by virtue of which only the essential details are displayed to the user (trivial or the non-essentials units are hidden).

- In java, abstraction is achieved by **interfaces** and **abstract classes**. We can achieve 100% abstraction using interfaces.

Abstract classes and Abstract methods :

- In **C++**, if a class has at least one **pure virtual** function, then the class becomes abstract. Unlike C++, in **Java**, a separate keywor **abstract** is used to **make** a **class abstract**.

- An **abstract method** is a method that is declared **without an implementation**.

- An abstract class **may or may not** have all abstract methods. Some of them can be concrete methods

- A method defined abstract must **always be redefined** in the subclass,thus making **overriding compulsory** OR either make **subclass itself abstract**.

- Any class that contains one or more abstract methods must also be declared with abstract keyword.

- There can be **no object** of an abstract class.That is, an abstract class **can not be directly instantiated** with the **new** operator.

- An abstract class **can have parametrized constructors** and **default constructor** is always present in an abstract class. It is called when an instance of a **inherited class** is created.

- Example:

```
// Java program to illustrate the concept of Abstraction

abstract class Shape  {

   String color;
   // these are abstract methods
   abstract double area();
   public abstract String toString();

   public Shape(String color) {   // abstract class can have constructor
      System.out.println("Shape constructor called");
      this.color = color;
   }

   public String getColor() {  // this is a concrete method
      return color;
   }
}
```

```
class Circle extends Shape  {
    double radius;
    public Circle(String color,double radius) {
        super(color);  // calling Shape constructor
        System.out.println("Circle constructor called");
        this.radius = radius;
    }
    double area() {
        return Math.PI * Math.pow(radius, 2);
    }
    public String toString() {
        return "Circle color is " + super.color +  "and area is : " + area();
    }
}
class Rectangle extends Shape{
    double length;
    double width;
    public Rectangle(String color,double length,double width) {
        super(color);    // calling Shape constructor
        System.out.println("Rectangle constructor called");
        this.length = length;
        this.width = width;
    }
    double area() {
        return length*width;
    }
    public String toString() {
        return "Rectangle color is " + super.color +  "and area is : " + area();
    }
}
public class Test  {
    public static void main(String[] args) {
        Shape s1 = new Circle("Red", 2.2);
        Shape s2 = new Rectangle("Yellow", 2, 4);
        System.out.println(s1.toString());
        System.out.println(s2.toString());
    }
}
```

Output:

```
Shape constructor called
Shape constructor called
Circle constructor called
Shape constructor called
Rectangle constructor called
Circle color is Red and area is: 15.205308
Rectanglr color is Yellow and area is: 8.0
```

- Like C++, in Java, an instance of an abstract class cannot be created, we can have references of abstract class type though.
- We can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated, but can only be inherited.
- Abstract classes can also have **final** methods (methods that cannot be overridden).

Interfaces in Java - Like a class, an interface can have methods and variables, but the methods declared in an interface are by **default abstract** (only method signature, no body).

- Interfaces specify **what** a **class must do** and **not how**. It is the **blueprint** of the class.
- An Interface is about **capabilities**. So it specifies a set of methods that the class has to implement.
- If a class implements an interface and **does not provide method bodies** for all functions specified in the interface, then the class **must be declared abstract**.
- A Java library example is, **Comparator Interface**.
- To declare an interface, use **interface** keyword.
- To implement interface use **implements** keyword.
- All the **methods** are **public** and **abstract**. And all the **fields** are **public**, **static**, and **final**.

Why do we use interface ?

- It is used to achieve **total abstraction (100%)**.
- By using interface, Java can achieve **multiple inheritance**. A class can implement more than one interface.
- It is also used to achieve **loose coupling**.
- Interfaces are used to implement abstraction.

Q. So the question arises why use interfaces when we have abstract classes?

A. The reason is, abstract classes may contain non-final variables, whereas variables in interface are **final**, **public** and **static**.

New features added in interfaces in JDK 8

1. Default Methods: After JDK 8, we can now add **default implementation** for **interface methods**. This default implementation has special use and does not affect the intention behind interfaces. Suppose we need to add a new function in an existing interface. Obviously the old code will not work as the classes have not implemented those new functions. So with the help of default implementation, **we will give a default body for the newly added functions**. Then the old codes will still work.

- Example of default methods

```
// An example to show that interfaces can have methods from JDK 1.8 onwards
interface In1 {
    final int a = 10;
    default void display()  {
        System.out.println("hello");
    }
}
// A class that implements the interface.
class TestClass implements In1 {
    // Driver Code
    public static void main (String[] args)  {
        TestClass t = new TestClass();
        t.display();
    }
}
```

Output:
hello

2. Static methods: Another feature is that we can now define **static methods in interfaces** which can be called independently without an object. **Note**: these methods are **not** inherited.

- Example of static methods in Interface

```
// An example to show that interfaces can have methods from JDK 1.8 onwards
interface In1 {
    final int a = 10;
    static void display() {
        System.out.println("hello");
    }
}
class TestClass implements In1 {
    public static void main (String[] args)  {
        In1.display();  // Calling with Interface's name
    }
}
```

Output:
hello

New features added in interfaces in JDK 9

1. Interfaces can now contain **Static methods**, **Private methods** and **Private Static method** as well.

Encapsulation vs Data Abstraction

- Encapsulation is **data hiding** (information hiding) while Abstraction is **detail hiding** (implementation hiding).
- While encapsulation **groups together** data and methods that act upon the data, data abstraction deals with **exposing the relevant functionalities** to the user and hiding the details of implementation.

ABSTRACTION	ENCAPSULATION
1. It is the process or method of gaining the information.	1. It is the process or method to contain the information.
2. Problems are solved at the design or interface level.	2. Problems are solved at the implementation level.
3. It is the method of hiding the unwanted information.	3. It is a method to hide the data in a single entity or unit along with a method to protect information from outside.
4. We can implement abstraction using **abstract class** and **interfaces**.	4. It can be implemented using by access modifier i.e. **private**, **protected** and **public**.
5. Implementation complexities are hidden using abstract classes and interfaces.	5. The data is hidden using methods of getters and setters.
6. The objects that help to perform abstraction are encapsulated.	6. Whereas the objects that result in encapsulation need not be abstracted.

Advantages of Abstraction

- It **reduces** the **complexity** of viewing the things.
- **Avoids code duplication** and **increases reusability**.
- Helps to **increase security** of an application or program as only relevant details are provided the user.