Constructor:

- A special type of method which is used to initialize an object when it is created.
- It has same name as the class name and does not have any return type (not even a void)

Why?

- Whenever an object of a class is created using the new keyword, java automatically calls a constructor to initialize the object.
- Without constructor we have to manually set the values of instance variables after creation of an object.

1. Default Constructor
2. Parameterized Constructor

TASK SOLUTION:

```
package que;

public class Person {
    //attributes
    String name;
    int age;

    //parameterized Constructor
    Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}


package que;

public class Student extends Person{
    //attribute
    String grade;

    //Constructor
```

```
    Student (String name, int age, String grade){
        super(name,age); // Calling the Parent class
constructor
        this.grade=grade;
    }

    void displayDetails(){
        System.out.println("Student: "+ name + ", Age : "+ age
+ ", Grade: "+ grade);
    }
}
```

```
package que;

public class Teacher extends Person{
    //attribute
    String subject;

    Teacher (String name, int age, String subject){
        super(name,age); // Calls the parent class constructor
        this.subject = subject;
    }

    void displayDetails(){
        System.out.println("Teacher: "+ name + ", age: "+ age
+ ", subject: "+ subject);
    }
}
```

```
package que;

public class Main {
    public static void main(String[] args) {
        Student gopal = new Student("Gopal",24,"A+");
        Teacher sharma = new Teacher("Sharma", 45,"Java");

        gopal.displayDetails();
        sharma.displayDetails();
    }
}
```

Abstraction:

Hiding implementation details and showing only the essential features.
TV remote(Only buttons are visible, not the internal circuits)

Encapsulation:

Wrapping of data(variables) and methods in a single unit(Class), restricting the direct access to data.

Ex. Medical capsule(The contents are enclosed/ encapsulated in a capsule).

Real-life analogy:

Car Interface:

1. Driver operates a car without knowing or understanding how the engine works(Abstraction)
2. The engine and other mechanical components are encapsulated within car's body (Encapsulation)

Abstract Class:

A class which cannot be instantiated and it may contain abstract methods(without body) and non-abstract method.

```
package abstractEx;

//Abstract class
abstract class Vehicle {

    //Abstract Method(without body/ without implementation)
    abstract void start();

    //Non-abstract method(with implementation)
    void stop(){
        System.out.println("Vehicle Stopped!!!");
```

```
    }
}
```

```
package abstractEx;

//Subclass car extends the abstract class vehicle
public class Car extends Vehicle{
    //Providing the implementation of abstract method
    void start(){
        System.out.println("Car Started!!!");
    }

}


package abstractEx;

public class Main {
    public static void main(String[] args) {
        // Creating an instance of a Car class
        Car car = new Car();

        // Call the start method from the car class
        car.start();

        //Calling the stop method(Inherited from vehicle
class)
        car.stop();

    }
}
```

- Interfaces

A completely abstract class with only abstract methods.

TASK: Create Animal interface

Define abstract method sound()

Create a class dog which will implement Animal

Provide the implementation of abstract method,

```java
package interfaceEx;

public interface Animal {

    //Abstract Method(No Impl)
    void sound();
}



package interfaceEx;

//Dog class implements the animal interfaces
public class Dog implements Animal{
  //providing the implementation of the abstract method
    @Override
    public void sound() {
        System.out.println("Dog Barks!!!!");
    }
}

package interfaceEx;

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();

        //Calling the sound method(defined in animal,
implemented in dog)
        myDog.sound();
    }
}
```

We can achieve loose coupling with interface.

Interfaces decouple the code by separating implementation details from definition.

Loose Coupling

Tight Coupling

Exceptions