Default Constructor:

- No Initialization is required

```java
package defaultExample;

public class Car {
    void display(){
        System.out.println("Car is running!!");
    }
}
```

```java
package defaultExample;

public class Test {
    public static void main(String[] args) {
        Car c = new Car();  // default constructor will get
called
        c.display();
    }
}
```

- To provide default values to fields

```java
package defaultExample;

public class Student {
    String name;
    int age;

    Student(){
        name = "Shreyas";
        age = 24;
    }

    void display(){
        System.out.println(name + " " + age);
    }
}
```

- Interfaces decouple the code by separating implementation details from definition.

Without interface(tightly coupled code)

```
package exOftightlyCoupled;

public class DieselEngine {
    void start(){
        System.out.println("Diesel Engine Started!!!");
    }
}
```

```
package exOftightlyCoupled;

public class Car {

    private DieselEngine engine = new DieselEngine(); //
Tightly coupled code


    void drive(){
        engine.start();
        System.out.println("Car is moving!!!!");
    }
}
/* Problem: If we want to switch to an electric engine, we
must modify the car.

 */
```

With Interface(Loosely Coupled Code)

```
package exOfLooselyCoupledCode;

public interface Engine {
    void start();
}
```

```java
package exOfLooselyCoupledCode;

public class DieselEngine implements Engine{

    @Override
    public void start() {
        System.out.println("Diesel Engine Started!!!");
    }
}
```

```java
package exOfLooselyCoupledCode;

public class ElectricEngine implements Engine{

    @Override
    public void start() {
        System.out.println("Electric Engine Started!!!");
    }
}
```

```java
package exOfLooselyCoupledCode;

public class Main {
    public static void main(String[] args) {
        Engine diesel = new DieselEngine();
        diesel.start();

        Engine electric = new ElectricEngine();
        electric.start();
    }
}
```

LooselyCoupledExample(After Session)

Interface paymentGateway -> void processPayment(doubleamount);

1. Class CreditCardPayment implements paymentGateway
   Public void processPayment(double amount){
   Sout(“Processing credit card payment of rupees” + amount)
   }
2. Class PhonePePayment ……
3. Class DebitCardPayment ……
4. Create main class

TASK:

Create BankAccount class which has private fields accountNumber, balance Add constructor to initialize the accountNumber and balance. Create methods deposit () and withdraw (), where: Deposit () adds to the balance. Withdraw () subtracts from the balance, but only if sufficient funds are available. In the main method, create an object of BankAccount, and demonstrate deposit and withdraw operations.

Exception:
An exception is an event which disrupts the flow of execution.

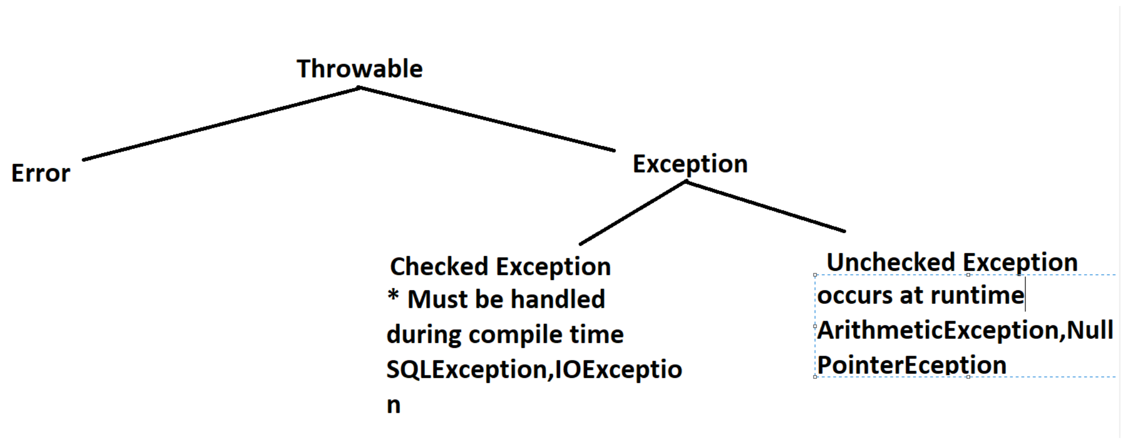
It represents the runtime issues,

1. Accessing an array index out of bounds
2. Division by zero
3. File handling errors

Difference between exception and error:

	Error	Exception
Definition	Serious issue beyond the application control	Issues caused by mistakes in the application code
Control	Not recoverable by the application	Recoverable using application code
Handling	Not Possible	Handled by try-catch block
Ex.	outOfMemoryError	NullPointerException

Exception Hierarchy: Throwable (Java SE 21 & JDK 21)



Types of Exception:

1. Built-in type exception

- Checked Exception and Unchecked Exception

2. User Defined Exception

We can create custom exceptions by extending exception class

* Methods to print exception information

  1. printStackTrace()
  2. toString()
  3. getMessage()