Interfaces:

A completely abstract class with only abstract methods.

package interfaceExample;  
  
public interface Animal {  
  
 //Abstract Methods(No Implementation)  
 void sound();  
}

package interfaceExample;  
  
//Dog class implements the animal interface  
public class Dog implements Animal{  
 //providing the implementation of the abstract method  
 public void sound(){  
 System.*out*.println("Dog Barks!!!");  
 }  
}

package interfaceExample;  
  
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 code by separating implementation details from definition.

Without interface: (tightly coupled code)

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

package exOfTightlyCoupling;  
  
public class Car {  
 private DieselEngine engine = new DieselEngine(); //Tightly coupled  
  
 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 { //Defines the contract  
 void start();  
}

package exOfLooselyCoupledCode;  
  
public class DieselEngine implements Engine{  
 @Override  
 public void start() {  
 System.*out*.println("Diesel Engine Started!!!");  
 }  
}

package exOfLooselyCoupledCode;  
  
public class ElectricEngine implements Engine{  
 @Override  
 public void start() {  
 System.*out*.println("Electric Engine Started!!!");  
 }  
}

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

Example of loosely coupled code:

package exOfPaymentGateway;  
  
public interface PaymentGateway {  
 void processPayment(double amount);  
}

package exOfPaymentGateway;  
  
public class CreditCardPayment implements PaymentGateway{  
  
 public void processPayment(double amount){  
 System.*out*.println("Processing credit card payment of rupees "+ amount);  
 }  
}

package exOfPaymentGateway;  
  
public class PhonePePayment implements PaymentGateway{  
 public void processPayment(double amount){  
 System.*out*.println("Processing PhonePe payment of rupees "+ amount);  
 }  
}

package exOfPaymentGateway;  
  
public class Main {  
 public static void main(String[] args) {  
 PaymentGateway payment1=new CreditCardPayment();  
 payment1.processPayment(100.0);  
  
 PaymentGateway payment2= new PhonePePayment();  
 payment2.processPayment(200.00);  
 }  
}

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.

Exceptions:

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

It represents the runtime issues,

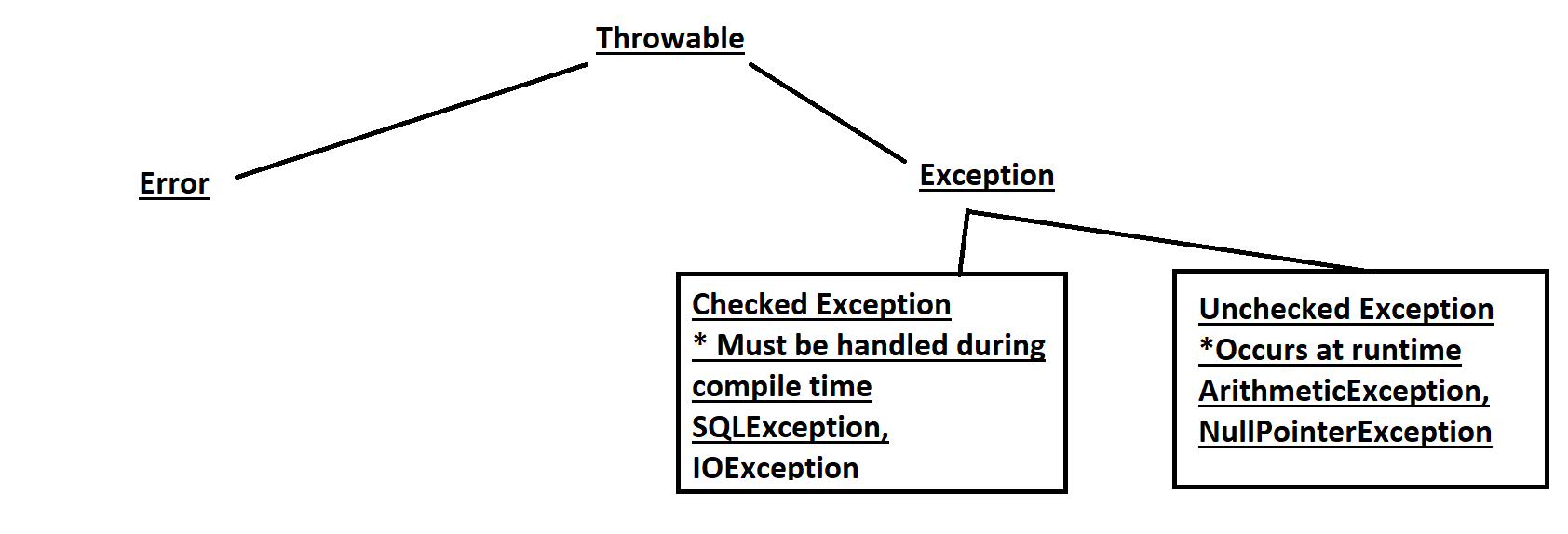
1. Accessing an array index out of bounds
2. Division by zero
3. File Handling errors

Difference between Exception and error.

|  |  |  |
| --- | --- | --- |
|  | Error | Exception |
| Definition | Serious issues 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 bock |
| Ex. | outOfMemoryError | NullPointerException |

[Overview (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/index.html)

Exception Hierarchy ([Throwable (Java SE 21 & JDK 21)](https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Throwable.html))



Types of Exceptions:

1. Built-in Exceptions

* Checked Exception and Unchecked Exception

1. User-defined Exceptions

* We can create custom exceptions by extending the exception class.
* These are useful when built-in exceptions don’t meet our requirements.

# Methods to print exception information

1.printStackTrace():

Prints the detailed information about the exception and its origin

2.toString()

Provides short info/desc of the exception, shows the class name and then message

3.getMessage()

Displays only the message associated with the exception.