Enumeration and Exception

Rawlabs Academy

What is Enumeration?

An **Enums** or **Enumerations** is a special *"class"* that represents a group of constants (unchangeable variables, like **final** variables).

Final variables

```
public class Level {
    public static final String LOW = "LOW";
    public static final String MEDIUM = "MEDIUM";
    public static final String HIGH = "HIGH";
}
```

Enum

```
public enum Level {
    LOW,
    MEDIUM,
    HIGH;
}
```

Why use Java Enums?

Enum was introduced to **replace the use of int constant**.

```
public class Size {
    public static final int SMALL = 1;
    public static final int MEDIUM = 2;
    public static final int LARGE = 3;
    public static final int EXTRA_LARGE = 4;
}
```

Can be simplify using enums.

```
public enum Size {
    SMALL, MEDIUM, LARGE, EXTRA_LARGE;
}
```

Java Enum - Basic Usage

```
public enum PizzaSize {
    SMALL, MEDIUM, LARGE, EXTRA_LARGE;
}

public class Main {
    public static void main(String[] args) {
        System.out.println(PizzaSize.MEDIUM);
        System.out.println(PizzaSize.LARGE);
    }
}
```

Java Enum in Switch Statement

```
public static void main(String[] args) {
    PizzaSize myPizza = PizzaSize.LARGE;
    switch(myPizza) {
        case PizzaSize.SMALL:
            System.out.println("I bought small pizza");
            break;
        case PizzaSize.MEDIUM:
            System.out.println("I bought medium pizza");
            break;
        case PizzaSize.LARGE:
            System.out.println("I bought large pizza");
            break;
    }
}
```

Methods of Java Enum

- `ordinal()` : returns the position of an enum constant
- `compareTo()` : compare the enum constants based on their ordinal value
- `toString()` : returns string representation
- `name()` : returns defined name in a string form
- `valueOf()` : takes a string and return an enum constant having the same string name
- `values()` : return an array of enum type containing all the enum constants

Enum with Predefined Value

```
public enum Color {
    RED("#F44336"),
    GREEN("#43A047"),
    BLUE("#0277BD");

    private final String hex;

    private Color(String hex) {
        this.hex = hex;
    }

    public String getHex() {
        return this.hex;
    }
}
```

Exception

What is Exceptions?

Exception is an abnormal condition.

In Java, an exception is an event that **disrupts** the normal flow of the program. It is object which is **thrown at runtime**.

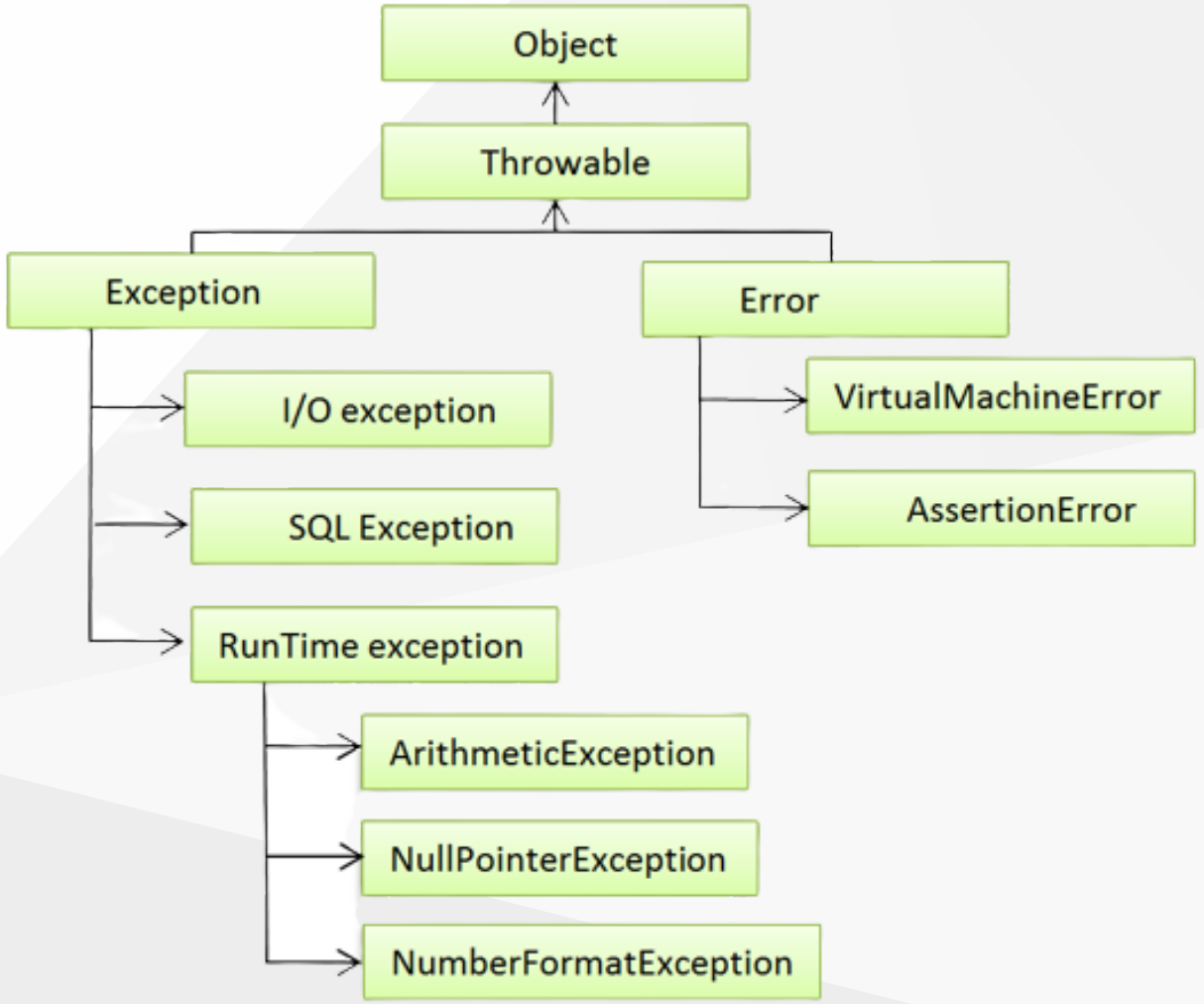
Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage : Maintain the normal flow of the application.

Exception Hierarchy

The `java.lang.Throwable` class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error



Types of Exception

Checked Exception

- The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions.
- Checked exceptions are **checked** at compile-time.

Example:

- `IOException`
- `SQLException`
- etc.

Unchecked Exception

- The classes that inherit the RuntimeException are known as unchecked exceptions.
- Unchecked exceptions are **not checked** at compile-time, but they are checked at runtime.

Example:

- `ArithmeticException`
- `NullPointerException`
- `ArrayIndexOutOfBoundsException`
- etc.

Error

Error is **irrecoverable**.

Example:

- `OutOfMemoryError`
- `VirtualMachineError`
- `AssertionError`
- etc.

Keywords

Keyword	Description
`try`	Specify a block where we should place an exception code
`catch`	Handle the exception. It must be preceded by try block which means we **can't** use catch block alone
`finally`	Execute the necessary code of the program. It is executed whether an exception is handled or not
`throw`	Throw an exception
`throws`	Declare exceptions, it specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Exception Handling Example

```
public static void main(String[] args) {
    int value = 0;
    try {
        System.out.println("This code should be throw an ArithmeticException");
        value = 100/0;
    } catch (ArithmeticException e) {
        System.out.println(e);
    }

    System.out.println("Code ended by ArithmeticException");
    System.out.println("Value still :: " + value);
}
```

Custom Exception

```
public class MyCustomException extends Exception {
    private String message;

    public MyCustomException(String message) {
        super();
        this.message = message;
    }
}
```

Cont...

```
public static void main(String[] args) {
    try {
        System.out.println("I will be throw a new MyCustomException");
        throw new MyCustomException("Hello world!!!");
        System.out.println("This code not executed.");
    } catch (MyCustomException e) {
        System.out.println("Exception message is :: ", e.getMessage());
    } finally {
        System.out.println("Finally block, will be execute in the end process");
    }
}
```

Task

Create a method to check **Boba drink** payment if the payment is less than the price then throw a custom exception. And make validation if the selected **Boba menu** does not match the enum, then throw a custom exception with a message.

Note: Take advantage of user input

```
Input Boba menu : Boba Tea
Input Size : REGULAR
Input payment : 120000

Your amount is less than price!
```

```
Input Boba menu : EXTRA_SMALL
Input Size: EXTRA_LARGE

Invalid pizza size!
Available size : [SMALL, REGULAR, LARGE]
```