Programming II

Object Oriented Programming (OOP)
Array List Class

The Array List Class:

- Similar to an array, an ArrayList allows object storage.
- Unlike an array, an Arraylist object:
  - Automatically expands when a new item is added.
  - Automatically shrinks when items are removed.
- Requires:

  import java.util.ArrayList;

Creating an ArrayList:

Declaration:

```
ArrayList<String> namelist = new ArrayList <String>();
```

- Notice the word String written inside angled brackets<>
- This specifies that the ArrayList can hold String objects.
- If we try to store any other type of objects in this ArrayList, an error will occur.
- For other types:
  - <Integer>
  - <Float>
  - <Double>
  - <Boolean>

Using an Array List:

- To populate the ArrayList, use the add method:
  - nameList.add("Ahmed");
  - namelist.add("Mahmoud");
- To get the current size, call the size method:
  - nameList.size(); // returns 2
- To access an item in an ArrayList, use the get method:
  - nameList.get(1);// 1 is the index of the element (zero based).

Using an ArrayList:

- The ArrayList class's toString method returns a string representing all ArrayList.
  - System.out.println(nameList);
  - System.out.println(nameList.toString());
- The ArrayList class's remove method that removes designated item from ArrayList
  - nameList.remove(1); // this statement removes the second item.

Using an ArrayList:

- The ArrayList class's add method with one argument adds new item at end of the ArrayList.
- To insert item at a location of choice, use the add method with two arguments.
  - nameList.add (1, "Ahmed"); // This statement inserts the String Ahmed at index 1. (Method Overloading)
- To replace an existing item, use the set method:
  - nameList.set(1, "Nour"); // This statement replaces Ahmed with Nour.

Example:

```
import java.util.ArrayList;
public class Main
{
    public static void main(String[] args) {
    ArrayList <Double> list = new ArrayList <Double>();
    list.add(7.7);
    list.add(10.0);
    list.add(1,8.0);
    System.out.println(list.toString());


    }
}
```

```
[7.7, 8.0, 10.0]
```

Example:

```
import java.util.ArrayList;
public class Main
{
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList <String>();
        list.add("Ahmed");
        list.add("Mohamed");
        list.add("Ali");
        System.out.println(list.size());
        System.out.println(list);
        System.out.println(list.get(2));
        list.add(1,"Nour");
        System.out.println(list);
        list.set(3,"Zaki");
        list.remove(2);
        System.out.println(list.size());
        System.out.println(list.toString());

    }
}
```

Output:

```
3
[Ahmed, Mohamed, Ali]
Ali
[Ahmed, Nour, Mohamed, Ali]
3
[Ahmed, Nour, Zaki]
```

Programming II

Object Oriented Programming (OOP)
Exception Handling

Handling Exceptions:

- An exception is an object that is generated as the result of an error or an unexpected event.
- Exception are said to have been “thrown”.
- It is the programmers responsibility to write code that detects and handles exceptions.
- Unhandled exceptions will crash a program.
- Java allows you to create exception handlers.

Example:

```
int x = 10, y = 0;
System.out.println (x/y);
```

Divide by 0 (Run Time)

Example:

```
public class Main
{
    public static void main(String[] args) {
        int x = 10, y = 0;
        System.out.println(x/y);
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:13)
```

Handling Exceptions:

- To handle an exception, you use a try statement.

```
try
{
   (try block statements...)
}
catch (ExceptionType ParameterName)
{
   (catch block statements...)
}
```

- First the keyword try indicates a block of code will be attempted.
- Keyword catch that can deal with the exception.

Example:

```
public class Main
{
    public static void main(String[] args) {
        try
        {int x = 10, y = 0;
          System.out.println(x/y);
            }
        catch(ArithmeticException e) {
            System.out.println("Error: Division by zero not allowed");
        }
        System.out.println("Lecture");

    }
}
```

```
Error: Division by zero not allowed
Lecture
```

Handling Multiple Exceptions:

- The code in the try block may be capable of throwing more than one type of exception.
- A catch clause needs to be written for each type of exception that could potentially be thrown.
- The Java will run the first compatible catch clause found.

Example:

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class Main
{
    public static void main(String[] args) {
        try
        { int x, y;
          Scanner input = new Scanner (System.in);
          System.out.println (" Enter x: ");
          x = input.nextInt();
          System.out.println (" Enter y: ");
          y = input.nextInt();
          System.out.println(x/y);
            }
        catch(ArithmeticException e) {
            System.out.println("Error: Division by zero not allowed");
        }

        catch(InputMismatchException e) {
            System.out.println("Error: please enter numeric values");
        }
    }
}
```

Outputs:

```
 Enter x:
4
 Enter y:
2
2
```

```
4
 Enter y:
0
Error: Division by zero not allowed
```

```
 Enter x:
4
 Enter y:
h
Error: please enter numeric values
```

Exception Classes:

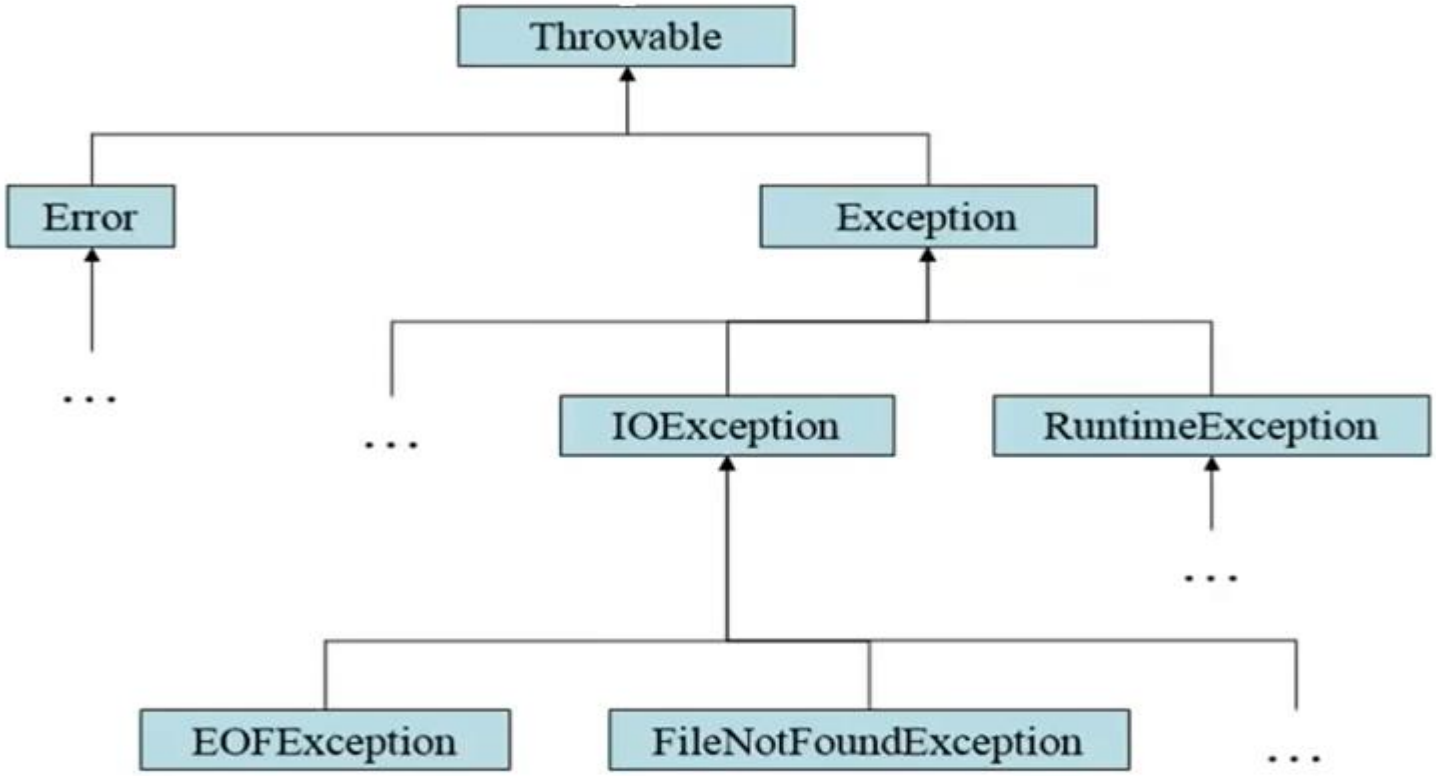
- An exception handler is a section of code that gracefully responds to exceptions.
- An exception is an object.
- Exception objects are created from exception classes in java.
- All of the exception classes in the hierarchy are derived from the throwable class.
- Error and Exception are derived from the throwable class.

Exception Classes:

FileNotFoundException is Exception

InputMisMatchException is Exception

AnyException is Exception



Example:

```
import java.util.Scanner;

public class Main
{
    public static void main(String[] args) {
        try
        { int x, y;
          Scanner input = new Scanner (System.in);
          System.out.println (" Enter x: ");
          x = input.nextInt();
          System.out.println (" Enter y: ");
          y = input.nextInt();
          System.out.println(x/y);
            }
        catch(Exception e)
        { System.out.println(e.getMessage());
    }
}
}
```

Returns a detailed message about the exception that has occurred

Output:

```
Enter x:
4
Enter y:
2
2
```

```
Enter x:
5
Enter y:
0
/ by zero
```

```
Enter x:
4
Enter y:
h
null
```