Name: **CYUZUZO Remy Patrick**
Reg.Number: **20RP00263**

Date: **01 May 2022**

JAVA RESEARCH

EXPLANATION & EXAMPLES

- **Exception Handling ()**

1) Try

The try statement allows you to define a block of code to be tested for errors while it is being executed.

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    }
  }
}
```

2) Catch

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

3) Finally

The finally keyword is used to execute code (used with exceptions - try...catch statements) no matter if there is an exception or not.

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
  }
}
```

4) Throw

The throw keyword is used to create a custom error. The throw statement is used together with an exception type.

Example:

Throw an exception if age is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class Main {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18
years old.");
    } else {
      System.out.println("Access granted - You are old enough!");
    }
 }

 public static void main(String[] args) {
   checkAge(15);
 }
}
```

5) Throws

The throws keyword indicates what exception type may be thrown by a method.

Example:

Throw an exception if age is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class Main {
  static void checkAge(int age) throws ArithmeticException {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18
years old.");
    } else {
      System.out.println("Access granted - You are old enough!");
    }
 }

 public static void main(String[] args) {
   checkAge(15);
 }
}
```

Let's see the difference between Throw and Throws in this table below:

THROW	**THROWS**
Used to throw an exception for a method	Used to indicate what exception type may be thrown by a method
Cannot throw multiple exceptions	Can declare multiple exceptions
Syntax: ▪ throw is followed by an object (new type) ▪ used inside the method	**Syntax:** ▪ throws is followed by a class ▪ and used with the method signature

- **Collection Classes ()**

1) Arraylists

The ArrayList class is a resizable array, which can be found in the java.util package. The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

2) Map

Map , represents a mapping between a key and a value. More specifically, a Java Map can store pairs of keys and values. Each key is linked to a specific value. Once stored in a Map , you can later look up the value using just the key. The Java Map interface is not a subtype of the Collection interface.

```
Map<String, String> map = new HashMap<>();

map.put("key1", "element 1");
map.put("key2", "element 2");
map.put("key3", "element 3");
```

3) Hashmap

Java HashMap is a class which is used to perform operations such as inserting, deleting and locating elements in a map. A HashMap however, store items in "key/value" pairs, and you can access them by an index of another type (e.g. a String).

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {
    HashMap<String, String> capitalCities = new HashMap<String, String>();
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

4) Lists

Java List is an ordered collection. Java List is an interface that extends Collection interface. Java List provides control over the position where you can insert an element. You can access elements by their index and also search elements in the list.Java List interface extends Collection interface. Collection interface externs Iterable interface.

Some of the most used List implementation classes are ArrayList, LinkedList, Vector, Stack, CopyOnWriteArrayList. AbstractList provides a skeletal implementation of the List interface to reduce the effort in implementing List.

- **int size():** to get the number of elements in the list.
- **boolean isEmpty():** to check if list is empty or not.
- **boolean contains(Object o):** Returns true if this list contains the specified element.
- **Iterator<E> iterator():** Returns an iterator over the elements in this list in proper sequence.
- **Object[] toArray():** Returns an array containing all of the elements in this list in proper sequence
- **boolean add(E e):** Appends the specified element to the end of this list.
- **boolean remove(Object o):** Removes the first occurrence of the specified element from this list.
- **boolean retainAll(Collection<?> c):** Retains only the elements in this list that are contained in the specified collection.

- **void clear():** Removes all the elements from the list.

5) Linked HashMap

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

Points to remember:

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.

```
import java.util.*;
class LinkedHashMap1{
 public static void main(String args[]){

  LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();

  hm.put(100,"Amit");
  hm.put(101,"Vijay");
  hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
 }
 }
}
```

- Nested Classes

  In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable. To access the inner class, create an object of the outer class, and then create an object of the inner class.

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

- Libraries

  Java is one of the most popular programming languages. Java provides a rich set of libraries, and its standard Java library is a very powerful that contains libraries such as java.lang, java.util, and java.math, etc. Java provides more than thousands of libraries except standard libraries.

  Some of the most useful and popular libraries are as follows:

  - Java Standard libraries
  - Apache Commons
  - Jackson
  - Maven

- Constants

Constant is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.

How to declare constant in Java?

In Java, to declare any variable as constant, we use static and final modifiers. It is also known as non-access modifiers. According to the Java naming convention the identifier name must be in capital letters.

```
import java.util.Scanner;
public class ConstantExample1
{
//declaring constant
private static final double PRICE=234.90;
public static void main(String[] args)
{
int unit;
double total_bill;
System.out.print("Enter the number of units you have used: ");
Scanner sc=new Scanner(System.in);
unit=sc.nextInt();
total_bill=PRICE*unit;
System.out.println("The total amount you have to deposit is: "+total_bill);
}
}
```

- Type Casting

In Java, type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

```
public class NarrowingTypeCastingExample
{
public static void main(String args[])
{
double d = 166.66;
//converting double data type into long data type
long l = (long)d;
//converting long data type into int data type
int i = (int)l;
System.out.println("Before conversion: "+d);
//fractional part lost
System.out.println("After conversion into long type: "+l);
//fractional part lost
System.out.println("After conversion into int type: "+i);
}
}
```

THE END