Automation Testing

Selenium with Java

Installation And Setup

- Java 8/11
- Setting Up Java Path
- Eclipse : Latest
- Maven
- TestNG
- Jenkins

Helpful Websites (Create Account)

- Oracle (To get JDK)
- Hacker Rank (To Practice Java)
- GitHub (Learn Source Code)
- LinkedIn
- FixMyCode (Under Dev)
- https://www.edx.org/

Introduction to Java programming

- JAVA was developed by Sun Microsystems Inc in 1991, later acquired by Oracle Corporation. It was developed by James Gosling and Patrick Naughton. It is a simple programming language. Writing, compiling and debugging a program is easy in java. It helps to create modular programs and reusable code.

Main Features of JAVA

- **Java is a platform independent language-** Compiler(javac) converts source code (.java file) to the byte code(.class file). As mentioned above, JVM executes the bytecode produced by compiler. This byte code can run on any platform such as Windows, Linux, Mac OS etc. Which means a program that is compiled on windows can run on Linux and vice-versa. Each operating system has different JVM, however the output they produce after execution of bytecode is same across all operating systems. That is why we call java as platform independent language
- **Secure**
- **Java is distributed**
- **Multithreading**

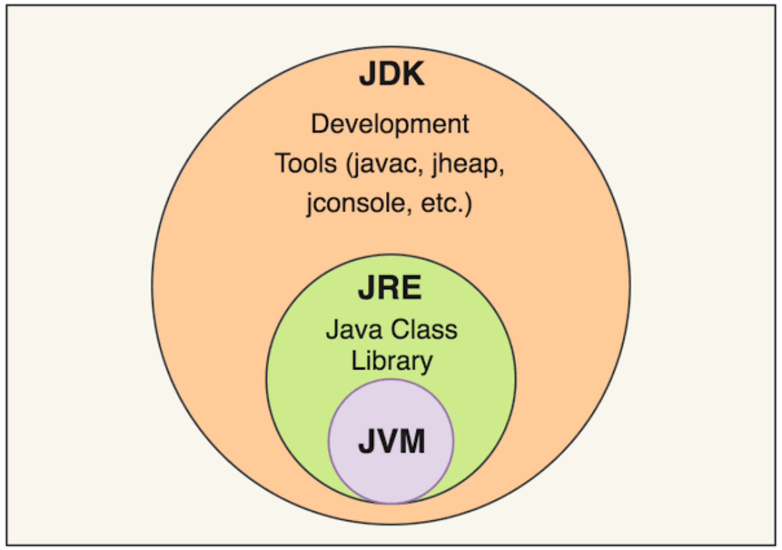
Main Features of JAVA

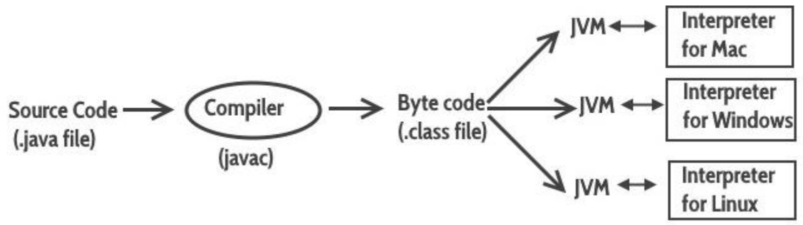
- **Java is an Object Oriented language-** Object oriented programming is a way of organizing programs as collection of objects, each of which represents an instance of a class.4 main concepts of Object Oriented programming are:

1. Abstraction
2. Encapsulation
3. Inheritance
4. Polymorphism

Java Key Components

- JDK (Java Development Kit)
- JRE (Java Runtime Environment)
- JVM (Java Virtual Machine)





First Java Program

- Compile Program Using cmd
- Use case of javac (javac FirstJavaProgram.java)
- Java command (java FirstJavaProgram)

Java Basics

1. **Variables in Java**: A variable is a name which is associated with a value that can be changed. For example when I write int i=10; here variable name is i which is associated with value 10, int is a data type that represents that this variable can hold integer values

Variables naming convention in java

1) Variables naming cannot contain white spaces, for example: int num ber = 100; is invalid

2) Variable name can begin with special characters such as $ and _

3) As per the java coding standards the variable name should begin with a lower case letter

There are three types of variables in Java.

1) Local variable 2) Static (or class) variable 3) Instance variable

Java Basics

1. **Java Data Types:** Data type defines the values that a variable can take, for example if a variable has int data type, it can only take integer values. In java we have two categories of data type: 1) Primitive data types 2) Non-primitive data types – Arrays and Strings are non-primitive data types

- **Java Operators:** An operator is a character that **represents an action**, for example + is an arithmetic operator that represents addition. **Types of Operator in Java**

- 1) Basic Arithmetic Operators
  2) Assignment Operators
  3) Auto-increment and Auto-decrement Operators
  4) Logical Operators

Java Basics

1. **If-else in Java**
2. **Switch-Case in Java**
3. **9. Java For loop**
4. **Java While loop**
5. **do-while loop in Java**
6. **Java Continue statement**
7. **Java Break statement**

Java Basics

1. **String**
2. **String Buffer**
3. **String Builder**
4. **Arrays**
5. **For each**
6. **Method/Functions**
7. **Method Input parameters**
8. **Method return type**

Java Basics

1. **Constructor**
2. **Create an Object of Class**
3. **Static Variable**
4. **Static Block**
5. **Packages**
6. **Method Overloading**
7. **Constructor Overloading**

OOPS Concept

1. **Encapsulation**
2. **Access Modifiers**
3. **Inheritance**
4. **Abstraction**
5. **Interface**
6. **Overloading**
7. **Overriding**
8. **Polymorphism**
9. **Exception Handling**

Encapsulation

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Other way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

- Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
- As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- Encapsulation can be achieved by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

Access Modifiers

1. **default**
2. **private**
3. **protected**
4. **public**

Default - The scope of this modifier is limited to the package only.

Private- The scope of private modifier is limited to the class only.

Protected- Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package

Public- The members, methods and classes that are declared public can be accessed from anywhere

Inheritance

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called **inheritance.**

Inheritance allows us to reuse of code, it improves reusability in your java application.

https://beginnersbook.com/2013/03/inheritance-in-java/

Types of inheritance in Java:

1. Single
2. Multiple
3. Multilevel
4. Hybrid

Abstraction(an idea but not having a concrete existence.)

A class that is declared using “**abstract**” keyword is known as abstract class. It can have abstract methods(methods without body) as well as concrete methods (regular methods with body). A normal class(non-abstract class) cannot have abstract methods**.**

An abstract class can not be **instantiated**, which means you are not allowed to create an **object** of it.

Why ?: https://beginnersbook.com/2013/05/java-abstract-class-method/

By making method abstract we made it compulsory to the child class to give implementation details to this method**.**

Rules:

1. A class derived from the abstract class must implement all those methods that are declared as abstract in the parent class**.**
2. Abstract class cannot be instantiated which means you cannot create the object of it so you can use the object of that child class to call non-abstract methods of parent class as well as implemented methods.
3. If a child does not implement all the abstract methods of abstract parent class, then the child class must need to be declared abstract as well.

Interface (100% Abstraction)

Interface looks like a class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract . Also, the variables declared in an interface are public, static & final by default.

Use of interface?- Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not allow you to extend more than one class, However you can implement more than one interfaces in your class.

- Without bothering about the implementation part, we can achieve the security of implementation
- In java, **multiple inheritance** is not allowed, however you can use interface to make use of it as you can implement more than one interface.
- Class implements interface but an interface extends another interface.

Abstract class vs interface in Java

https://beginnersbook.com/2013/05/abstract-class-vs-interface-in-java/

Overloading and Overriding

Overloading- Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

Overriding- Declaring a method in **sub class** which is already present in **parent class** is known as method overriding. Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class. In this case the method in parent class is called overridden method and the method in child class is called overriding method.

Rules of method overriding in Java

- Argument list: The argument list of overriding method (method of child class) must match the Overridden method(the method of parent class). The data types of the arguments and their sequence should exactly match.

- Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class. For e.g. if the Access Modifier of parent class method is public then the overriding method (child class method) cannot have private, protected and default Access modifier,because all of these three access modifiers are more restrictive than public.

Polymorphism (feature that allows us to perform a single action in different ways)

Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways. For example, lets say we have a class Animal that has a method sound(). Since this is a generic class so we can’t give it a implementation like: Roar, Meow, Oink etc.

Static or Compile Time (Overloading)- Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.

Here the method demo() is overloaded 3 times: first method has 1 int parameter, second method has 2 int parameters and third one is having double parameter. Which method is to be called is determined by the arguments we pass while calling methods. This happens at compile time so this type of polymorphism is known as compile time polymorphism.

Dynamic or Run Time (Overriding)-Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism.

Java Exception handling

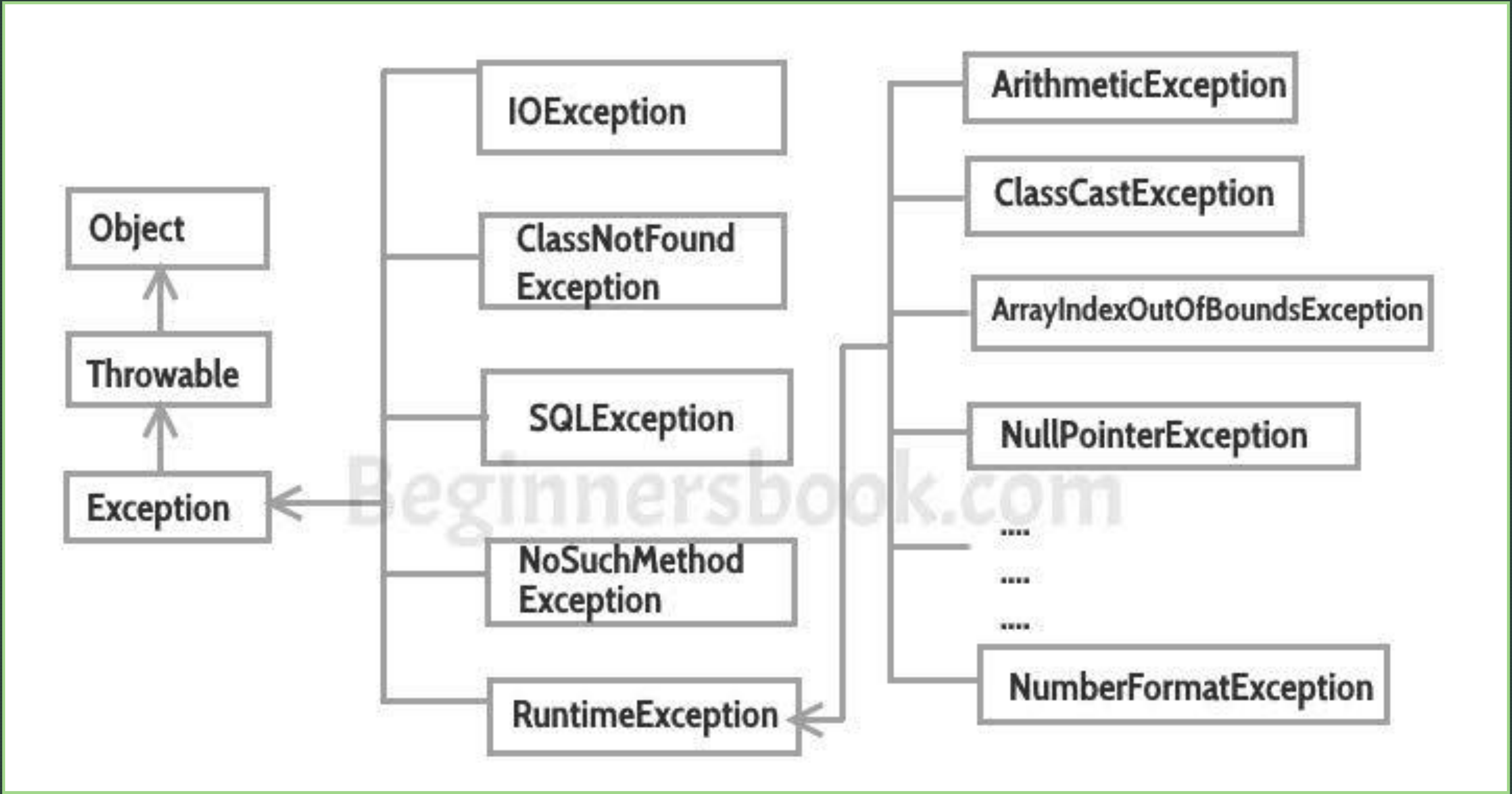
What is an exception? An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message.

Why an exception occurs? There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exception Handling: If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.

Difference between error and exception

- **Errors** indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.
- **Exceptions** are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions


Object
Throwable
Exception
IOException
ClassNotFound Exception
SQLException
NoSuchMethod Exception
RuntimeException
ArithmeticException
ClassCastException
ArrayIndexOutOfBoundsException
NullPointerException
....
....
....
NumberFormatException

Java Exception handling

Types of exceptions

Checked exceptions
Unchecked exceptions

Checked exceptions-All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, you will get compilation error. For example, SQLException, IOException, ClassNotFoundException etc.

Unchecked Exceptions-Runtime Exceptions are also known as Unchecked Exceptions. These exceptions are not checked at compile-time so compiler does not check whether the programmer has handled them or not but it’s the responsibility of the programmer to handle these exceptions and provide a safe exit. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Compiler will never force you to catch such exception or force you to declare it in the method using throws keyword.

Java Exception handling-Try Catch in Java

```
try
{
    //statements that may cause an exception
}
catch (exception(type) e(object))
{
    //error handling code
}
```

Java Exception handling-Finally Block

A **finally block** contains all the crucial statements that must be executed whether exception occurs or not. The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.

```
try {

    //Statements that may cause an exception

}catch {

   //Handling exception

}finally {

   //Statements to be executed

}
```

A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.

The statements present in the **finally block** execute even if the try block contains control transfer statements like return, break or continue

Whenever System.exit() gets called in try block then **Finally block** doesn’t execute

Java Exception handling-Throw

We can define our own set of conditions or rules and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException when we divide number by 5, or any other numbers, what we need to do is just set the condition and throw any exception using throw keyword. Throw keyword can also be used for throwing custom exceptions.

https://beginnersbook.com/2013/04/user-defined-exception-in-java/

```
throw new exception_class("error message");
```

In this program we are checking the Student age if the student age<12 and weight <40 then our program should return that the student is not eligible for registration.

```
public class ThrowExample {

  static void checkEligibilty(int stuage, int stuweight){

    if(stuage<12 && stuweight<40) {

      throw new ArithmeticException("Student is not eligible for registration");

    }

    else {

      System.out.println("Student Entry is Valid!!");

    }

  }
```