Java Programming

Basic -   
Current – 20+

Most Popular – Java 8 and Java 16 ( Spring 2+), Java 17 ( Spring boot 3+)

JDK vs JRE -> Java Development Kit and Java Run time Environment

Features :

Simple, Platform Independent, Robust Secure, Multi threading

IDE – Intellij IDEA Community Version, Eclipse , STS

Setting environment variables

We ran sample program ( HelloWorld) -> Create a sample program and run

public class HelloWorld {  
  
 String name;  
   
 String firstName;  
 String lastName;  
   
 Integer age;  
 String address;  
  
 static String *city*;  
 static String *state*;  
  
 static Integer *zipCode*;  
 static {  
 *city* = "Illinio";  
 *state* = "Ohio";  
 *zipCode* = 10;  
 }  
  
 //Naming Variable  
   
  
 public static void main(String[] args) {  
 System.*out*.println("Hello World, Welcome Java programming!!");  
 //Creating an object  
 HelloWorld obj1 = new HelloWorld();  
 obj1.name = "Rahul";  
 obj1.age = 12;  
 obj1.address = "sample address";  
 System.*out*.println("Name : " +obj1.name);  
 System.*out*.println("Age : " +obj1.age);  
 System.*out*.println("Address : " +obj1.address);  
  
 //Invoking a method using object  
 obj1.display();  
 obj1.display2();  
  
 //invoking a static method  
 HelloWorld.*display3*();  
 System.*out*.println("City : " +HelloWorld.*city* + ", State " +HelloWorld.*state*+ ", Zip code : "+HelloWorld.*zipCode*);  
 }  
  
 public static void main(String[] args, Integer a) {  
 System.*out*.println("Hello World, Welcome Java programming!!");  
 }  
  
 //static methods are associated with classes  
 public static void main(String[] args, boolean flag) {  
 System.*out*.println("Hello World, Welcome Java programming!!");  
 }  
  
 //non-static method are associated with objects  
 public void display() {  
 System.*out*.println("I am inside display method");  
 }  
  
 //non-static method are associated with objects  
 public void display2() {  
 System.*out*.println("I am inside display method 2");  
 }  
  
 public static void display3() {  
 System.*out*.println("I am inside display method 3 and it is static method");  
 }  
  
   
  
}

Access Modifiers and Local Variables  
package training.basics;  
  
public class AccessModifier {  
  
  
 private int data = 50;  
  
 public static void main(String[] args) {  
  
 //Java Variable - Local Variables, Class Variables( Static variables) , Instance variables ( non static variables)  
 String localVariable1 = "I am local variable";  
 System.*out*.println("Local variable : " +localVariable1);  
  
 AccessModifier accessModifier = new AccessModifier();  
 accessModifier.displayMethod1();  
 System.*out*.println("Accessing private variable : " +accessModifier.data);  
  
 //Access Modifier - public, private , protected and default  
  
 }  
  
 public void displayMethod1() {  
  
 //Local variable  
 Integer var1 = 100;  
 System.*out*.println("inside display method 1 , var1 : " +var1);  
 }  
  
}

Constructors

public class Constructor {  
 String name;  
 Integer age;  
  
 public Constructor() {  
 System.*out*.println("I am inside constructor");  
 this.name = "Rahul";  
 this.age = 10;  
 }  
  
 public Constructor(String name, Integer age) {  
 System.*out*.println("I am inside parameterized constructor");  
 this.name = name;  
 this.age = age;  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("i am inside main");  
 Constructor constructor = new Constructor();  
 System.*out*.println("Name : " + constructor.name + ", age : " + constructor.age);  
 Constructor constructor1 = new Constructor("Mohan", 12);  
 System.*out*.println("Parameterized constructor values : Name - " + constructor1.name + ", Age : " + constructor1.age);  
 }  
}

Inheritance

package training.inheritance;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
  
public class Department {  
  
 String departmentName;  
 String departmentCode;  
  
 public List<String> getEmployee() {  
  
 System.*out*.println("i am inside getEmployee method of Department class");  
 List<String> employees = new ArrayList<>();  
 employees.add("emp1");  
 employees.add("emp2");  
 employees.add("emp3");  
 employees.add("emp4");  
 return employees;  
 }  
  
  
}

package training.inheritance;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Finance extends Department {  
  
 public static void main(String[] args) {  
 Department department = new Department();  
 department.departmentCode = "DEPARMENT\_CODE";  
 department.departmentName = "DEPARTMENT";  
  
 List<String> departments = department.getEmployee();  
  
 System.*out*.println("Department list : " + departments);  
  
 Finance finance = new Finance();  
 List<String> financeEmployees = finance.getEmployee();  
 System.*out*.println("Finance list : " + financeEmployees);  
  
 Department department1 = new Finance();  
 department1.departmentName = "FINANCE";  
 department1.departmentCode = "FINANCE\_CODE";  
 System.*out*.println("Department Name " +department1.departmentName);  
 System.*out*.println("Department Code " +department1.departmentName);  
 System.*out*.println("Calling getEmployee method - " +department1.getEmployee());  
 }  
  
 @Override  
 public List<String> getEmployee() {  
 System.*out*.println("i am inside getEmployee method of Finance class");  
 List<String> employees = new ArrayList<>();  
 employees.add("emp3");  
 employees.add("emp4");  
 employees.add("emp5");  
 employees.add("emp6");  
 return employees;  
  
 }  
  
}

**Polymorphism**

1. Method Overloading ( Static Polymorphism/Compile Polymorphism )
2. Method Overridding ( Dynamic Polymorphism/Runtime Polymorphism )

**Method Overloading (Static Polymorphism/Compile Polymorphism )**

package training.polymorphism;  
  
public class MethodOverloadingExample {  
  
 public Integer sum(int a, int b) {  
 System.*out*.println("i am inside sum method with 2 arguments");  
 Integer result = a + b;  
 return result;  
 }  
  
 public Integer sum(int a, int b, int c) {  
 System.*out*.println("i am inside sum method with 3 arguments");  
 Integer result = a + b + c;  
 return result;  
 }  
  
 public Integer sum(int a, int b, int c, int d) {  
 System.*out*.println("i am inside sum method with 4 arguments");  
 Integer result = a + b + c + d;  
 return result;  
 }  
  
 public Double sum(double a, double b) {  
 System.*out*.println("i am inside sum method with 2 arguments having double value");  
 Double result = a + b;  
 return result;  
 }  
  
 public static void main(String[] args) {  
 MethodOverloadingExample obj = new MethodOverloadingExample();  
  
 //Other way of calling a method and printing the result  
 System.*out*.println("Sum of 2 numbers : " +obj.sum(3,5));  
  
 System.*out*.println("Sum of 3 numbers : " +obj.sum(3,5,6));  
  
 System.*out*.println("Sum of 4 numbers : " +obj.sum(3,5,6, 6));  
  
 System.*out*.println("Sum of 4 numbers : " +obj.sum(2.5,3.5));  
 }  
  
  
  
  
  
}

**Method Overridding (Dynamic Polymorphism/Runtime Polymorphism )**

**Abstraction : Two way to achieve abstraction in Java**

1. **Abstract class**
2. **Interface**

**Abstract class:**

It can have abstract and non-abstract methods.

It can have constructor and static methods.

Abstract classes are partially abstract.

**Interface:**

It can have static and final variables.

Interfaces are fully abstract.

It can have only abstract methods until Java 7 Version

Post java 8, interface can also have default and static methods as well.

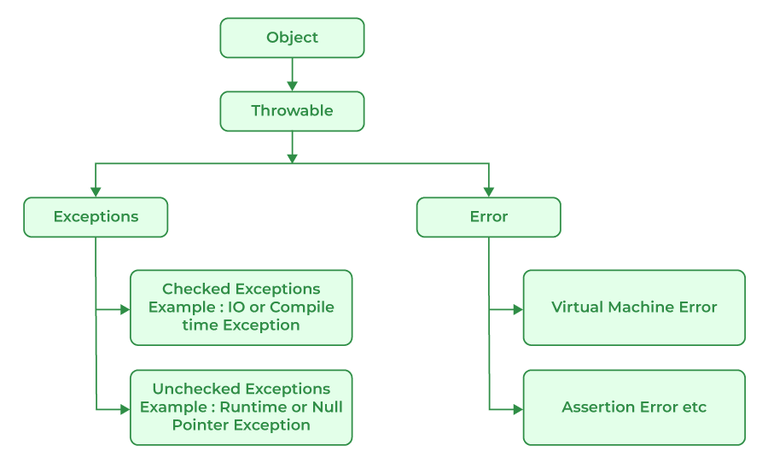
Wrapper Classes

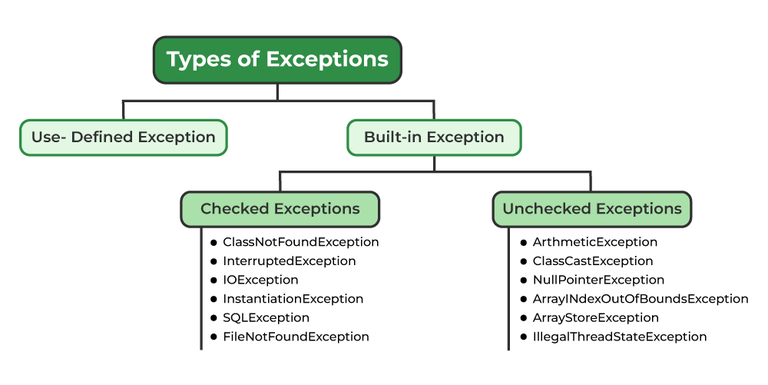
A table with different types of wrapper class

Description automatically generated

| **StringBuffer Class** | **StringBuilder Class** |
| --- | --- |
| StringBuffer is present in Java. | StringBuilder was introduced in Java 5. |
| StringBuffer is synchronized. This means that multiple threads cannot call the methods of StringBuffer simultaneously. | StringBuilder is asynchronized. This means that multiple threads can call the methods of StringBuilder simultaneously. |
| Due to synchronization, StringBuffer is called a thread safe class. | Due to its asynchronous nature, StringBuilder is not a thread safe class. |
| Due to synchronization, StringBuffer is lot slower than StringBuilder. | Since there is no preliminary check for multiple threads, StringBuilder is a lot faster than StringBuffer. |

Exceptions in Java





Built in Exceptions:

1. Checked Exception
2. Unchecked Exception

Different ways to throw an Exception

1. Using try catch block
2. Using throws key word which we need to use at method level