JAVA

Features of Java:

It was developed by James Gosling in 1995,later acquired by Oracle Corporation.

Before Java,its Name was OAK.

Java is an high Level .**ROBUST--Memory Management.**Object -Oriented and Secure Language

Uses:

Mobile applications

Destop application

Web apps

Web servers and App Server

Games,etc..

JDK_JAVA DEVELOPMENT KIT:

The JDK is an Software Development is used to Develop java apps and applet

The JDK is Physically exists.

Contains:

JRE-Java Runtime Enviroinment

A Private Java Virtual Machine (JVM)

An Intrepreter/loader(JAVA)

A compiler (JAVAC)

JRE,JVM,JIT:

JRE-Java Runtime Environment is a set of Software Tools,used for developing JAVA apps.

Used to provide Runtime Environment

JVM-Java Virtrual Machine acts as a run-time engine to run Java apps

Interprets the bytecode line by line and then executes to produce machine Code.

JIT- Just In Time compiler is used to increase efficiency of Intrepreter

It compiles the entire Bytecode and changes it to native code

Features of Java:

Simple

Object oriented

Platform Independent

Robust

Portable

High Performance

Secure

Simple:

Java is easy to learn and its syntax is quite,Simple,clean and easy to understand.

Java excludes operator overloading.Pointers..etc

Object Oriented:

In Java Excludes is an object which has some data and behaviour.

Platform Independent:

on compilation java program is compiled into bytecode.

This bytecode is platform independent and can be run on any machine

The Bytecode format also Provide Security

Robust:

Uses strong memory management.

There are exception handling and the type checking mechanism in java

Portable:

Java is Portable because it facilities you to carry the java bytecode to any platform.

High Performance:

Java enables high Performance with the use of just in the compiler

Secure:

Java Program always runs in java runtime environment with the most null with the system OS,hence it is more Secure

DataTypes:

The primitive datatype are the predefined datatypes provided by the java Programming language

There are eight Primitive Types.They are Byte ,short,int,long,float,double,boolean and char.

The Process of converting the primitive type to corresponding wrapper class object automatically is known as autoboxing

A wrapper class in java is used to convert a primitive data type to an object and object to an Primitive Type

Even the Primitive datatype are used to storing primary datatypes,data structures such as array List and vectors store objects.

The process of automatically converting an objects of a wrapper class to its corresponding Primitive type is Known as Unboxing.

String Handling Functions:

Method	**Description**	**Return Value**
Char At(int)	Find the character at given index	char
Length ()	Finds the length of given string	int
Compare To (String)	Compare two strings	int
Compare to ignore Case (String)	Compare a two strings, ignore case	int
Case(string)	Concatenates the object string with argument string	string
Contains(string)	Checks whether a string contains sub-string	boolean
Is empty	Checks whether a string is empty or not	boolean

OOP---Object oriented Programming

Class

A class is a blueprint for creating objects (instances).

Fields:

Model and year, which hold the state of the object.

Constructor:

Car(String model, int year) initializes the values of model and year when a new Car object is created.

Method:

displayInfo() prints the details of the car to the console.

Code:

```
class Car {

  String model;
  int year;

  public Car(String model, int year) {
    this.model = model;
    this.year = year;
  }
  public void displayInfo() {
    System.out.println("Car Model: " + model);
    System.out.println("Year: " + year);
  }
}

public class Main {

  public static void main(String[] args) {
    // Create an object (instance) of the Car class
    Car myCar = new Car("Toyota Corolla", 2020);
    myCar.displayInfo();     }
}
```

Object:

An **object** in Java is an instance of a class. It is created based on the blueprint provided by a class and can hold data (fields) and perform actions (methods).

Code:

```
class Person {

  String name;
  int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }

  public void introduce() {
    System.out.println("Hi, my name is " + name + " and I am " + age + " years old.");
  }
}

public class Main {
    public static void main(String[] args) {
        Person person1 = new Person("Alice", 30);  // person1 is an object of class
Person



        person1.introduce();  // Output: Hi, my name is Alice and I am 30 years old.



        Person person2 = new Person("Bob", 25);
        person2.introduce();    }
}
```

Inheritance:

Inheritance is one of the core principles of Object-Oriented Programming (OOP). It allows a new class (subclass or child class) to inherit the properties and behaviors (fields and methods) of an existing class (superclass or parent class).

Code:

```
class Animal {

    String name;
    int age;

    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void speak() {
        System.out.println("The animal makes a sound.");
    }
}

class Dog extends Animal {

    public Dog(String name, int age) {
        super(name, age);
    }

    @Override
    public void speak() {
        System.out.println(name + " says Woof!");
    }
}

public class Main {
    public static void main(String[] args) {

        Dog myDog = new Dog("Buddy", 3);

        System.out.println("Dog's Name: " + myDog.name);
```

```
            System.out.println("Dog's Age: " + myDog.age);


            myDog.speak();
        }
    }
```

Encapsulation:

It refers to the concept of bundling the data (fields) and methods (functions) that operate on the data into a single unit (a class).

Encapsulation also restricts direct access to some of an object's components, which is typically done by making fields private and providing public getter and setter methods to access and modify those fields.

Code:

```
class BankAccount {



  private String accountHolder;
  private double balance;



  public BankAccount(String accountHolder, double balance) {
    this.accountHolder = accountHolder;
    this.balance = balance;
  }
```

```
public String getAccountHolder() {
    return accountHolder;
}

public void setAccountHolder(String accountHolder) {
    this.accountHolder = accountHolder;
}



public double getBalance() {
    return balance;
}

public void setBalance(double balance) {

    if (balance >= 0) {
        this.balance = balance;
    } else {
        System.out.println("Balance cannot be negative.");
    }
}

  public void deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }



    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: $" + amount);
        } else {
            System.out.println("Invalid withdrawal amount or insufficient funds.");
        }
    }

    public void displayAccountInfo() {
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Balance: $" + balance);
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("John Doe", 500.0);
        account.displayAccountInfo();

        account.deposit(200.0);



        account.withdraw(100.0);

        System.out.println("Updated Balance: $" + account.getBalance());

        // Try setting a negative balance (this will be rejected)
        account.setBalance(-100.0);
    }
}
```

Polymorsphism:

Polymorphism is one of the core concepts in Object-Oriented Programming (OOP) that allows objects of different types to be treated as objects of a common superclass.

1. **Compile-time Polymorphism** (also known as **Method Overloading**)

2. **Runtime Polymorphism** (also known as **Method Overriding**)

Code:

```
class Calculator {

    public int add(int a, int b) {
        return a + b;
    }


    public int add(int a, int b, int c) {
        return a + b + c;
    }

    public double add(double a, double b) {
        return a + b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        System.out.println("Sum of 3 and 4: " + calc.add(3, 4));
        System.out.println("Sum of 3, 4, and 5: " + calc.add(3, 4, 5));
System.out.println("Sum of 2.5 and 3.7: " + calc.add(2.5, 3.7));     }
}----------------------------
```
Method Overloading

-------------------Method Overriding-------------------------------

```
class Animal {
    public void sound() {
        System.out.println("The animal makes a sound.");
    }
}

class Dog extends Animal {

    @Override
    public void sound() {
        System.out.println("The dog barks.");
    }
}

class Cat extends Animal {

    @Override
    public void sound() {
        System.out.println("The cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {

        Animal myDog = new Dog();  // Dog is treated as Animal (Polymorphism)
        Animal myCat = new Cat();  // Cat is treated as Animal (Polymorphism)



        myDog.sound();
  myCat.sound();     }
}
```

Abstraction:

Abstraction is another key principle of Object-Oriented Programming (OOP) in Java. It refers to the concept of hiding the implementation details and showing only the essential features of an object

Abstraction allows you to focus on what an object does, rather than how it does it. In Java, abstraction is achieved using abstract classes and interfaces.

Code:

```
abstract class Shape {
    public abstract void draw();



    public void description() {
        System.out.println("This is a shape.");
    }
}



class Circle extends Shape {
    public void draw() {
        System.out.println("Drawing a circle.");
    }
}

class Rectangle extends Shape {

    public void draw() {
        System.out.println("Drawing a rectangle.");
    }
}

public class Main {
    public static void main(String[] args) {

        Shape circle = new Circle();
        Shape rectangle = new Rectangle();

        circle.draw();
        rectangle.draw();

        circle.description();
    }
}
```

Static:

The static keyword in Java is used for memory management primarily. When a field or method is declared as static, it means the field or method belongs to the **class** rather than instances of the class (objects).

Static variables: Shared by all instances of the class.

Static methods: Can be called without creating an instance of the class.

Static blocks: Used for static initialization.

Code:

```
class Counter {
  static int count = 0;

  Counter() {
    count++;  // Increment count whenever a new object is created
  }

  static void displayCount() {
    System.out.println("Count: " + count);
  }
}

public class Main {
  public static void main(String[] args) {

    Counter c1 = new Counter();
    Counter c2 = new Counter();
    Counter c3 = new Counter();
```

```
        Counter.displayCount();
    }
}
```

Final:

The final keyword is used to declare constants, prevent method overriding, and prevent inheritance.

Final variable: Once assigned a value, it cannot be changed (constant).

Final method: Cannot be overridden by subclasses.

Final class: Cannot be inherited by any other class.

Code:

```
final class Vehicle {

    final String type = "Car";

    final void displayType() {
        System.out.println("Vehicle type: " + type);
    }
}

public class Main {
    public static void main(String[] args) {

        Vehicle vehicle = new Vehicle();
        vehicle.displayType();



    }
}
```

Exception :

An abnormal event that occurs during program execution that disputes the noramal flow of the program instruction

An error condition that changes the normal of flow of control in a program

A single that some unexpected condition has occured in the Program

Handling Exception:

Exception mechanism is built arround the throw and catch program paradiagram

“To throw “ Exception has occured

“To catch” Deal with an exception

If exception is not caught it is Propagated to the call stack until a handler is found

Type of Exceptions:

All exceptions in java are objects of the throwable class .

Unchecked Exception

Checked Exception

Unchecked Exceptions:

}

Exceptions derived from error and runtime exception classes.

Are usually irrecorable and not handled explicitly are not checked by the compiler

Checked Exceptions:

Are Exceptions derived from the Exception class excluding RuntimeException Classes

Must be handled explicitly

Are checked by the compiler.

Both checked and unchecked Exception are thrown and caught.

New Exception are created by extending the Exception class or its Subclass.

Try-catch()-Finaly

```
Try
{


}
catch (Exception e)
{


}
catch(Exception2 ee)
{


}
```

```
                    catch (Exception3 eee)

                    {


                    }

                    finally{


                    }
```

Unhandled exception:

```
Public class grading System{

Public static void (string args[ ])

GradingSystem gs=new GradingSystem();

gs.checkGrades();





}


Void checkgrades(){

int []grades=null;


For (int i=0; i<grades.length;i++){




}
}

}
```

Exception handling:

Exceptional conditional sometimes occur in properly written code and must be handled

exception handling in java is build arround the “throw and catch” paradiagram

If a method can’t handle the exception,it can propagate the exception to the call stack,

All exception are objects are throwable class

Only checked exception must be handled explicitly

Try-catch()-finally statement is an mechanism of java.

Trowable statement manually create an exception or rethrows an exception.

Throws statement allows propagation of exception to the call stack.

Throw and throws

```
void validate() throws exception 3{
Try
{

Throw new exception3();

}
Catch (Exceptiona1 ex)
{
}
Catch(exception2 ex2){



}
Catch(exception3 ex3){
Throw new Exception 3(ex)
}
}






}
```

Try Catch Exceptions

```
Package Exceptions;
Public class tryCatchExample{

Public static void main(String args[]){

TryCatch.divide(12,0);
System.out.println("after exception")
}
}

Package Exceptions;

Public static void divide(int number1,int number2)
{
Int result;
Try{
Result=number1/number2;
Sytem.out.println(result);

}
Catch(Arthematic Exception ex){

System.out.println("cannot divide the number by zero"+ ex)

}

}
}
```

Multiple catch Exception

```
          Package Exception;
```

```
Public class multiple catch{

Public static void (String args[]){


Try{


Int[] value =new int[5];

Value[9]=25/0;

}

Catch (arithmaticException arithmaticxception)

{


System.out.println("Arithmatic exception");

}

Catch(arrayIndexout-ofBoundsException exception){

System.out.println("array index out of bounds");

}

System.out.println("Program contains after exception");

}


}
```

---------------------------------Finally Block----------------------------------

```
Package Exceptions;

Public class Finally{

Public static void main(String args[])
{

Int value1,value2,result;

Try{

Value 1=12;

Value 2=0;

Result=value 1/value 2;

System.out.println("cannot be executed")


}

Catch(array-index out of boundException Exception){


System.out.println("array index out of bound");

}

Finally{

System.out.println("Finally block will be executed");

}


}


}
```