1. History of Java & Java Buzzwords

Ans: Java is a high-level, object-oriented, platform-independent programming language.

- Developed by **James Gosling** and his team at *Sun Microsystems* in 1991 (initially called **Oak**).
- In **1995**, it was renamed **Java** and made publicly available.
- In **2009**, Oracle Corporation acquired Sun Microsystems, and Java is now maintained by Oracle.

Main Features (Java Buzzwords): Java became popular because of its simplicity, portability, and robustness, making it one of the most widely used programming languages.

- **Simple** – No pointers, no operator overloading.
- **Object-Oriented** – Focus on objects and reusability.
- **Platform-Independent** – Runs on JVM, supports "Write Once, Run Anywhere (WORA)".
- **Secure** – No direct memory access, bytecode verification.
- **Robust** – Strong memory management, exception handling.
- **Multithreaded** – Supports multiple tasks simultaneously.
- **Portable** – Bytecode can run on any platform.
- **Dynamic** – Supports runtime linking of classes.

---

2. Operators in Java

Ans: Operators are symbols that perform operations on variables and values. Operators in Java make expressions concise and help in performing calculations, decisions, and assignments.

Java provides:

- **Arithmetic** (+ - * / %)
- **Relational** (> < == != >= <=)
- **Logical** (&& || !)
- **Assignment** (= += -= *= /=)
- **Unary** (++ --)
- **Ternary** (condition ? true : false)

Program

```
public class Operators {
    public static void main(String[] args) {
        int a = 10, b = 5;
        System.out.println("a + b = " + (a + b));
        System.out.println("a > b = " + (a > b));
   System.out.println("(a > 5 && b < 10) = " + (a > 5 && b < 10));
```

```
    }
}
```

Output

```
a + b = 15
a > b = true
(a > 5 && b < 10) = true
```

---

3. Datatypes in Java

Ans:

- **Primitive types**: byte, short, int, long, float, double, char, boolean.
- **Non-primitive types**: String, Arrays, Classes, Objects.
  Datatypes define the **type of data a variable can hold** and the **size of memory allocated**.
- Datatypes ensure type safety and efficient memory usage in Java.
- Datatypes ensure type safety and efficient memory usage in Java.

Program

```
public class Datatypes {
    public static void main(String[] args) {
        int age = 20;
        double salary = 55000.5;
        char grade = 'A';
        boolean pass = true;
        String name = "Java";
System.out.println(age + ", " + salary + ", " + grade + ", " + pass + ", " +
name);
    }
}
```

Output

```
20, 55000.5, A, true, Java
```

---

4. Variables and Literals

Ans: Variables store data, and literals represent constant values used in expressions.

- **Variable**: Named memory location that stores data.
- **Literals**: Fixed values assigned to variables.

Types of literals

- Integer (10)
- Floating-point (3.14)
- Character ('A')

- String ("Hello")
- Boolean (true/false)

Program

```
public class VariablesLiterals {
    public static void main(String[] args) {
        int number = 100;
        double pi = 3.1416;
        char grade = 'A';
        String name = "Java";
        boolean status = true;
System.out.println(number + ", " + pi + ", " + grade + ", " + name + ", " +
status);
    }
}
```

---

5. Classes and Objects

Ans: Objects represent real-world entities, while classes provide the structure for those objects.

- **Class**: Blueprint that defines variables (data) and methods (functions).
- **Object**: Instance of a class that uses memory.

Program

```
class Car {
    String brand;
    int speed;
    void display() {
        System.out.println("Brand: " + brand + ", Speed: " + speed);
    }
}
public class ClassObject {
    public static void main(String[] args) {
        Car c1 = new Car();
        c1.brand = "Toyota";
        c1.speed = 120;
        c1.display();
    }
}
```

Output

```
Brand: Toyota, Speed: 120
```

---

6. this Keyword

Ans: this keyword is useful in constructors and methods to refer to the current object.

- Refers to **current object**.
- Used to **resolve conflicts** between local variables and instance variables.

Program

```
class Student {
    String name;
    Student(String name) {
        this.name = name;
    }
    void display() {
        System.out.println("Name: " + name);
    }
}
public class ThisKeyword {
    public static void main(String[] args) {
        Student s = new Student("John");
        s.display();
    }
}
```

---

7. Constructor Overloading

Ans: Constructor overloading provides flexibility in object creation.

- **Constructor** → Special method used to initialize objects.
- **Overloading** → Multiple constructors with different parameter lists.

Program

```
class Person {
    String name;
    int age;
    Person(String n) {
        name = n;
        age = 0;
    }
    Person(String n, int a) {
        name = n;
        age = a;
    }
    void display() {
        System.out.println(name + " " + age);
    }
}
public class ConstructorOverloading {
    public static void main(String[] args) {
        Person p1 = new Person("Alice");
        Person p2 = new Person("Bob", 25);
        p1.display();
        p2.display();
    }
}
```

---

8. Method Overloading

Ans: Method overloading increases code readability and flexibility.

- Same method name, different parameter list (number/type).
- Provides **compile-time polymorphism**.

Program

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double b) {
        return a + b;
    }
}
public class MethodOverloadingDemo {
    public static void main(String[] args) {
        Calculator c = new Calculator();
        System.out.println("Sum int: " + c.add(5, 10));
        System.out.println("Sum double: " + c.add(5.5, 10.5));
    }
}
```

---

9. Control Statements

Ans: Control statements decide program flow based on conditions and loops. Used to control the flow of execution in Java.

- **Decision-making**: if, if-else, switch
- **Looping**: for, while, do-while

- **Jump**: break, continue

Program

```
public class ControlStatements {
    public static void main(String[] args) {
        int x = 10;
        if(x > 5)
            System.out.println("x is greater than 5");
        for(int i=1; i<=3; i++)
            System.out.println("i = " + i);
        switch(x) {
            case 10: System.out.println("x is 10"); break;
            default: System.out.println("Other value");
        }
    }
}
```

---

10. Inheritance

Ans: Inheritance promotes **code reusability** and implements **OOP hierarchy**. Mechanism of acquiring properties of one class into another. Keyword → extends. Supports **single, multilevel, and hierarchical inheritance**.

Program

```
class Animal {
    void eat() {
        System.out.println("Animal eats food");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }
}
public class Inheritance {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();
        d.bark();
    }}
```

Output: Animal eats food

Dog barks