CERTIFIED QUIZ

Introduction to Java

Java is a high-level, class-based, object-oriented programming language that is designed to have as few implementation dependencies as possible. It was originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun's Java platform. Java is platform-independent, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Structure of a Java Program

A Java program typically includes the following key components:
- Package Declaration: A way to organize Java classes into namespaces.
- Class Declaration: The main building block in Java which contains methods and variables.
- Main Method: The entry point of any Java application, represented as `public static void main(String[] args)`.

Example:
java
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}

Data Types in Java

Java provides a rich set of data types, categorized into Primitive Types and Reference Types:

- Primitive Types: Include `int`, `float`, `double`, `boolean`, `char`, `byte`, `short`, and `long`. These are predefined by the language and named by a keyword.
- Reference Types: Include objects, arrays, and interfaces. These types store references to the actual data.

Example:
java
int age = 30;
float salary = 50000.5f;
boolean isJavaFun = true;
char grade = 'A';

Variables and Constants

Variables are containers for storing data values. In Java, variables must be declared before they are used.

Syntax:
java
int age = 25;
float pi = 3.14f;

Constants: In Java, constants are declared using the `final` keyword. Once a constant is assigned, its value cannot be changed.

Example:

java
final int MAX_VALUE = 100;

Operators in Java

Java supports a wide variety of operators:

- Arithmetic Operators: `+`, `-`, `*`, `/`, `%` (for addition, subtraction, multiplication, division, and modulus).
- Relational Operators: `==`, `!=`, `>`, `<`, `>=`, `<=` (for comparison).
- Logical Operators: `&&`, `||`, `!` (AND, OR, NOT).
- Bitwise Operators: `&`, `|`, `^`, `~` (for manipulating individual bits).
- Assignment Operators: `=`, `+=`, `-=`, `*=`, `/=` (for assigning and updating values).

Example:
java
int a = 5;
int b = 3;
int result = a + b; // result = 8

Control Structures

Control structures allow Java programs to make decisions and repeat certain blocks of code:

- if-else: Executes a block of code based on a condition.
java
if (age >= 18) {
System.out.println("You are an adult.");
} else {
System.out.println("You are not an adult.");
}

- switch-case: Allows selection among multiple options based on an expression.
java
switch (grade) {
case 'A':
System.out.println("Excellent");
break;
case 'B':
System.out.println("Good");
break;
default:
System.out.println("Invalid grade");
break;
}

- Loops: Java supports various looping mechanisms like `for`, `while`, and `do-while`.
- `for` loop:
java
for (int i = 0; i < 5; i++) {
System.out.println(i);
}

- `while` loop:
java

```
int i = 0;
while (i < 5) {
System.out.println(i);
i++;
}
```

- `do-while` loop (executes at least once):
java
```
int i = 0;
do {
System.out.println(i);
i++;
} while (i < 5);
```

Object-Oriented Programming (OOP) in Java

Java is a fully object-oriented language. The key concepts of OOP in Java include:

- Classes: Define the blueprint for an object.
- Objects: Instances of classes that have states (attributes) and behaviors (methods).
- Encapsulation: Wrapping data and code into a single unit, ensuring controlled access through access modifiers like `private`, `protected`, and `public`.
- Inheritance: A mechanism where one class can inherit properties and behaviors from another class using the `extends` keyword.
- Polymorphism: The ability to use a single interface to represent different types of objects. It comes in two forms: compile-time (method overloading) and runtime (method overriding).
- Abstraction: Hides implementation details and exposes only the essential functionalities using abstract classes and interfaces.

Example:
java
```
class Animal {
void sound() {
System.out.println("Animal makes a sound");
}
}

class Dog extends Animal {
void sound() {
System.out.println("Dog barks");
}
}

public class Main {
public static void main(String[] args) {
Animal myDog = new Dog();
myDog.sound(); // Output: Dog barks
}
}
```

Methods in Java

A method in Java is a block of code that performs a specific task. Methods promote code reusability.

Syntax:
java

```
return_type methodName(parameters) {
// method body
return value;
}
```

Example:
java
```
public int addNumbers(int a, int b) {
return a + b;
}
```

Constructors

A constructor in Java is a special type of method that is invoked when an object is instantiated. It is used to initialize the object's state.

- Constructors have the same name as the class.
- They do not have a return type (not even `void`).
- Java provides a default constructor if no constructor is defined.

Example:
java
```
class Person {
String name;

// Constructor
public Person(String name) {
this.name = name;
}
}

public class Main {
public static void main(String[] args) {
Person p = new Person("John");
System.out.println(p.name); // Output: John
}
}
```

Exception Handling in Java

Java provides a robust mechanism for handling runtime errors through exceptions. Exception handling ensures that the normal flow of the application is maintained.

The basic structure includes:
- try: Block of code that may generate an exception.
- catch: Block that catches and handles the exception.
- finally: Block that is executed regardless of whether an exception occurred.

Example:
java
```
try {
int result = 10 / 0; // This will throw an exception
} catch (ArithmeticException e) {
System.out.println("Cannot divide by zero");
} finally {
```

System.out.println("This will always execute");
}

Collections Framework

The Java Collections Framework provides a set of classes and interfaces for managing groups of objects. It includes:

- List: An ordered collection (e.g., `ArrayList`, `LinkedList`).
java
List list = new ArrayList<>();
list.add("Java");
list.add("Python");

- Set: A collection that contains no duplicate elements (e.g., `HashSet`, `TreeSet`).
java
Set set = new HashSet<>();
set.add("Java");
set.add("Java"); // Duplicate won't be added

- Map: A collection of key-value pairs (e.g., `HashMap`, `TreeMap`).
java
Map map = new HashMap<>();
map.put("John", 25);
map.put("Jane", 30);