Unit I

Question Bank

1. List features of following Programming Paradigms

->

a. **Procedural :**
   - It is command driven language
   - It works through the state of the machine
   - Its semantics are quite though
   - It returns only restricted data types and allows values
   - Overall efficiency is very high
   - It is not suitable for time critical applications

b. **Object Oriented**
   - **Encapsulation**: Data and code are bundled together into objects.
   - **Abstraction**: Objects expose only their public interface, hiding their internal implementation.
   - **Inheritance**: Objects can inherit properties and behaviors from other objects.
   - **Polymorphism**: Objects can be treated in different ways depending on their type.

c. **Functional**
   - Pure functions: Functions have no side effects and always return the same output for the same input.
   - Immutability: Data is immutable, meaning that it cannot be changed once it is created.
   - Recursion: Functions can call themselves.

d. **Logic**
   - Declarative programming: Programs are written as a set of facts and rules, rather than as a sequence of steps.
   - Backtracking: The program can try different solutions to a problem until it finds one that works.
   - Unification: The program can match different patterns of data.

2. What is Language Standardization?

-> In computer programming, a programming language Standardization is a documentation that defines a programming language, so that users and implementors can agree on what programs in that language mean. Specifications are typically detailed and formal, and primarily used by implementors, with users referring to them in case of ambiguity; the C++ specification is frequently cited by users, for instance, due to the complexity. documentation includes a programming language reference, which is intended expressly for users, and a programming language rationale, which explains why the specification is written as it is; these are typically more informal than a specification

3. What are the criteria's to select syntax?

-> Readability: The syntax should be easy to read and understand. This means that the syntax should be consistent, regular, and concise.

Expressiveness: The syntax should be able to express complex ideas in a clear and concise way.

Conciseness: The syntax should be concise, meaning that it should be able to express ideas with a minimal amount of code.

Consistency: The syntax should be consistent throughout the language. This means that similar concepts should be expressed in a similar way.

Regularity: The syntax should be regular, meaning that there should be few exceptions to the rules of the language.

Orthogonality: The features of the language should be orthogonal, meaning that they should be independent and can be combined in any way without unexpected results.

Completeness: The language should be complete, meaning that it should be able to express all of the ideas that a programmer might need to express.

Extensibility: The language should be extensible, meaning that it should be possible to add new features to the language without breaking existing code.

Portability: The language should be portable, meaning that code written in the language should be able to run on different platforms without modification.

Efficiency: The language should be efficient, meaning that code written in the language should run quickly and use memory efficiently.

Safety: The language should be safe, meaning that it should be difficult to write code that contains errors.

Security: The language should be secure, meaning that it should be difficult to write code that can be exploited by attackers.

4. List the components of Programming language.

> The components of programming language are divided into two types : **Syntax and Semantics**

Syntax is the set of rules that define how a programming language should be written. It includes the rules for forming valid expressions, statements, and declarations.

Semantics is the meaning of a programming language. It defines what each expression, statement, and declaration does.

The various components includes :

Data types: Data types define the different kinds of data that can be represented in the language.

Control flow statements: Control flow statements allow programmers to control the order in which statements are executed.

Functions: Functions allow programmers to group together related statements and reuse them throughout their code.

Libraries: Libraries provide pre-written code that programmers can use to perform common tasks.

Keywords: Keywords are reserved words that have special meaning to the compiler or interpreter.

Identifiers: Identifiers are names that programmers give to variables, functions, classes, and other objects.

Literals: Literals are constant values, such as numbers, strings, and characters.

Operators: Operators are symbols that perform operations on data, such as addition, subtraction, multiplication, and division.

Expressions: Expressions are combinations of operators and operands that evaluate to a value.

Statements: Statements are instructions that tell the computer what to do.

Declarations: Declarations are used to introduce new variables, functions, classes, and other objects.

5. Explain the process of translation

->

The translation process in Java programming is a two-step process: **Compilation and Interpretation**

Compilation: The Java compiler translates Java source code into Java bytecode. Java bytecode is a platform-independent code that can be run on any platform that has a Java Virtual Machine (JVM) installed.

Interpretation: The JVM interprets Java bytecode and executes it on the underlying hardware platform.

The following is a more detailed explanation of each step:

Compilation : The Java compiler is responsible for translating Java source code into Java bytecode. The compiler performs a number of tasks during compilation, including:

Lexical analysis: The compiler breaks the source code into tokens, which are the smallest units of code that have meaning to the compiler.

Syntactic analysis: The compiler parses the tokens and checks to make sure that the source code is syntactically correct.

Semantic analysis: The compiler checks to make sure that the source code is semantically correct, meaning that it makes sense in the context of the Java language.

Code generation: The compiler generates Java bytecode from the source code.

Interpretation :The JVM is responsible for interpreting Java bytecode and executing it on the underlying hardware platform. The JVM performs a number of tasks during interpretation, including:

Loading: The JVM loads the Java bytecode into memory.

Verification: The JVM verifies that the Java bytecode is valid and that it does not contain any security vulnerabilities.

Execution: The JVM executes the Java bytecode by converting it to machine code and executing it on the underlying hardware platform.

The translation process in Java programming is designed to be efficient and secure. The compiler performs a number of checks to make sure that the source code is syntactically and semantically correct. The JVM also performs a number of checks to make sure that the Java bytecode is valid and that it does not contain any security vulnerabilities.

6. What are the types of composite data types?

-> Def : Composite data types are data types that have one or more fields dynamically linked to fields in another data type. Composite data types are useful for creating a single data type that references information in more than one data source or that references more than one table or other structure in a single data source.

Example of **composite data type** is <u>**array**</u>, **lists** and **objects**.

Example of array : int a[ ] = {1,2,3,4,5};

In above example, is composite because an array-of-int value is comprised of some number of element values chosen from the int type. Using composite data types, we can manage multiple pieces of related data as a single datum. Record and objects are composite data type. Functions are technically a type of object and are therefore considered composite data

7. Write a short note on Scalar data type

-> Scalar data types in Java programming is that they are data types that can only represent a single value. They are the simplest type of data type in Java and are used to store a wide variety of data, such as numbers, characters, and Boolean values.

Scalar data types are also used to implement more complex data structures, such as arrays and objects. For example, an array is a data structure that can store a collection of scalar data types of the same type. An object is a data structure that can contain both scalar data types and other objects.

Here are some scalar data types :

- byte: A byte can store a value from -128 to 127.
- short: A short can store a value from -32,768 to 32,767.
- int: An int can store a value from -2,147,483,648 to 2,147,483,647.
- long: A long can store a value from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
- float: A float can store a decimal value.
- double: A double can store a decimal value with greater precision than a float.
- char: A char can store a single character.
- boolean: A boolean can store a true or false value.

8. Explain properties of elementary data types

-> Elementary data types in Java programming are also known as **primitive data types**. They are the simplest data types in Java and are **used to store a wide variety of data, such as numbers, characters, and Boolean values.**

Size: The size of an elementary data type is the number of bytes it occupies in memory. The size of an elementary data type varies depending on the data type. For example, a byte occupies 1 byte in memory, while a long occupies 8 bytes in memory.

Range: The range of an elementary data type is the set of values that it can store. The range of an elementary data type varies depending on the data type.

Default value: The default value of an elementary data type is the value that is assigned to a variable of that data type if no value is explicitly assigned. The default value of an elementary data type varies depending on the data type. For example, the default value of a byte is 0, while the default value of a boolean is false.

9. Why data types are required in programming languages

-> Data types are required in programming languages because they tell the compiler or interpreter how to represent and manipulate the data that is being used in the program

Accuracy: Data types help to ensure that data is represented accurately in memory. For example, if a variable is declared as a byte, the compiler will make sure that the variable is only assigned values that can be represented in a byte. This helps to prevent errors such as overflows, which can occur when a variable is assigned a value that is too large for its data type.

Efficiency: Data types also help to improve the efficiency of programs. For example, the compiler can generate more efficient code for operations on variables of known data types. This is because the compiler can make assumptions about the size and range of values that a variable can hold.

Security: Data types can also help to improve the security of programs. For example, if a variable is declared as a private integer, the compiler will prevent other parts of the program from modifying the variable. This helps to prevent errors and security vulnerabilities.

10. Explain role of a programming language.

-> A programming language is a formal language that specifies a set of instructions for a computer to perform specific tasks. It is used to write software programs and applications, and to control and manipulate computer systems.

Programming languages play an essential role in many aspects of our lives. They are used to develop the software that powers our computers, smartphones, and other devices. They are also used to create websites, web applications, and mobile apps. In addition, programming languages are used in a wide variety of industries, including finance, healthcare, and manufacturing.

Here are some of the **key roles** of a programming language:

Communication between humans and computers: Programming languages allow humans to communicate with computers in a way that computers can understand. By writing code in a programming language, programmers can tell computers what to do and how to do it.

Creation of software: Programming languages are used to create software programs and applications. Software programs are sets of instructions that tell computers how to perform specific tasks. Software applications are programs that are designed to be used by users, such as web browsers, word processors, and video games.

Control of computer systems: Programming languages can be used to control and manipulate computer systems. This includes tasks such as managing files and directories, running programs, and configuring networks.

Automation of tasks: Programming languages can be used to automate tasks. This means that programmers can write code to perform tasks that would otherwise be done manually. This can save time and improve efficiency.

Problem-solving: Programming languages can be used to solve problems. Programmers can write code to solve a wide variety of problems, such as mathematical problems, scientific problems, and engineering problems.