Characteristics of Java: Java is ___

- Distributed: Distributed computing involves several computers working together on a network. Java is designed to make distributed computing easy. Since networking capability is inherently integrated into Java, writing network programs is like sending and receiving data to and from a file.
- Interpreted: You need an interpreter to run Java programs. The programs are compiled into the Java Virtual Machine code called bytecode. The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the Java Virtual Machine (JVM).
- Robust: Java compilers can detect many problems that would first show up at execution time in other languages. Java has eliminated certain types of error-prone programming constructs found in other languages. Java has a runtime exception-handling feature to provide programming support for robustness.
- Secure: Java implements several security mechanisms to protect your system against harm caused by stray programs.
- Architecture-Neutral: Write once, run anywhere, with a Java Virtual Machine (JVM), you can write one program that will run on any platform.
- Portable: Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.
- Performance: Java's performance Because Java is architecture neutral, Java programs are portable. They can be run on any platform without being recompiled.
- Multithreaded: Multithread programming is smoothly integrated in Java, whereas in other languages you must call procedures specific to the operating system to enable multithreading.
- Dynamic: Java was designed to adapt to an evolving environment. New code can be loaded on the fly without recompilation. There is no need for developers to create, and for users to install, major new software versions. New features can be incorporated transparently as needed.
- Simple, Object-Oriented: *WRITE ANYTHING*.

MAIN: public static void main(String[] args){}

Circle[] circleArray = new Circle[3];

An array of objects is an array of reference variables.

Generics Restriction:

class Sample <T extends Number> {T data; …;

T[] method(T[] a) {return a;} }

Visibility:

- private: Inside class only

~ No modifier (default): Classes in Package

protected: Classes in package and subclasses inside or outside the package.

+ public: Every Where.

Important Stuff:

Boolean equals(): compares contents of two objects. However, must be overridden for custom classes.

final: usable with classes, local variables, data members, methods (prevent method from being overridden by a subclass), Class {class cannot be extended}

Relationships:

Final (class): represented by stereotype {leaf}.

Final (data member): written in capital letters.

Abstract class/method: represented in italic font or using stereotype <<abstract>>

Interface: represented by stereotype <<interface>>

- Generalization "inheritance":
  - "Child is a person", "is a" - "kind-of" relationship.
  - Between classes or classes and interfaces
- Dependency:
  - Import B;
  - Class A {method1(B b){….} }
  - Class A depends on the existence of class b since it receives a B object in the constructor.
- Aggregation:
  - NEXT PAGE 'D

- Aggregation:
    - Class A has a list of Class B.
    - However, Class A can work even when having no B objects in the list unlike Dependency where object A wouldn't have been created without a B object.
    - Redundant form of Association: receiving a reference of B object/s in Class A.
    - class Asset { ... }
      class Player {
      List assets;
      public void AddAsset(Asset newlyPurchasedAsset) {
      assets.Add(newlyPurchasedAssest); ... } }
- Composition
    - Class A is the owner of Class B objects
    - Existence of the B objects depends on the existence of the class A object as it is the owners of the created B's
    - public class Piece { ...}
      public class Player
      { Piece piece = new Piece(); /*Player owns the responsibility of creating the Piece*/ }

Polymorphism:

Polymorphism means that a variable of a supertype can refer to a subtype object.

- GeoObj go =new Circle();