What is Java -

Java is a High-level language. A High-level language cannot be run directly on a computer; first it has to be translated into a machine language by a program called Compiler. In Java source code is compiled to intermediate byte code, which is then interpreted and executed by a JVM

Java is Object-Oriented

Java is a platform

A virtual machine is a software application that simulates a computer but hides the underlying OS and hardware from the programs that interact with it.

Java's byte-codes are portable between platforms without recompiling.

JVM -

In early java versions the JVM was simply an interpreter for java byte codes. Most java programs would execute slowly because the JVM would interrupt in executing one byte-code at a time. Today's JVMs typically execute byte-codes using a combination of interpretation and just-in-time (JIT) compilation.

JVM does garbage collection

JVM provides security. Things that are running inside the JVM are sand-boxed. They have the safety because the programs running inside virtual machine is flawless likely to disrupt the user OS or corrupt the data files if errors occur

Java is strongly typed, types must be declared and it cannot change.

Class -

A class is a general idea

A class provides blueprints for creating an Object

A group of related methods and data

Object -

An instance of a Class

An object is the real thing

Classes have members - Data and Method.

Encapsulation -

Packaging of complex functionality to make it easy to use in an application

It helps in hiding the true nature of data and methods except what is absolutely necessary to expose

Control statements -

Conditional - if-else, switch

Iterative - for, while, do-while

Jumping - return, break, continue

In switch, if we don't add break statements at the end of each case, then it falls through the code and executes all below it.

Inheritance -

Classes do not stand alone, they are related

Inheritance is an Is-A relationship

All classes inherit from Object

Composition -

Composition is a Has-A relationship

Also known as Aggregation

We need ways to model real relationship between things in-order to be able to reuse code.

Favour Composition -

Inheritance is very difficult to maintain, especially when we make changes to a base class, those changes ripple down and cause breakage in all the sub-classes

Almost all inheritance can be rewritten as composition

Interface -

It specifies what the contract is, what the method signature looks like for some class that going to implement it

No default implementation details unlike inheritance

Generic -

A way to write code independent of type

Class level - House<T> type as a variable

Method level - public <U> void printUsingCatridge(U cartridge) type as a parameter

Type Erasure -

Java Generics aren't real, they are erased after being compiled and are not present in the byte code

Only compiler uses generic to determine whether something is wrong or right

We cannot check the type of a generic class used because it doesn't exist at runtime

Exception -

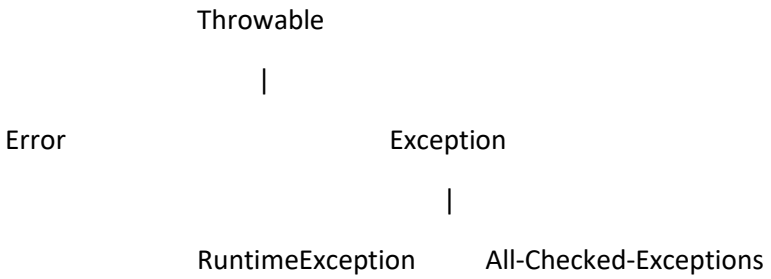
Exceptions are the basic ways to indicate there are errors and to handle that errors

Exceptions that are not dealt with crash the application

All Exceptions inherit from a common type called Throwable

Checked - sub-classes of Exception and these has to be handled

Unchecked - sub-classes of Error or RuntimeException (indeed a sub-class of Exception) and these need not to be handled



Always throw IllegalArgumentException (a RuntimeException) when the value of argument passed is not proper.

finally blocks are basically used to prevent resource leaking.

Exception handling best practices -

Catch exception as specific as possible

Do not catch errors

Do not catch exceptions that you aren't going to do something with

Collection -

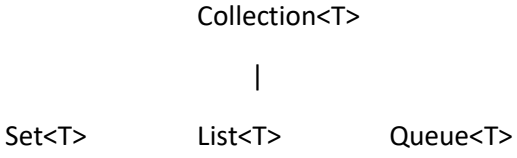
Collection is a group of objects

List - ordered and sequential, can have duplicates

Set - no duplicates, must be able to compare

Queue - ordered list, designed for processing

Map - key-value pair, key must be unique



Enumeration -

Enumeration is an exact listing of all elements of a set

Enumerations are used in place of string constants, when constraining input is beneficial, also can be used as a Singleton

Comparing enums is type safe, also it limits the input

Enumerations are iterable too, by using the values() method

Enumerations are classes, it has members as well as constructor

Enumeration constructors have to be private so that we can't call them from outside, otherwise we would be able to create more types

Java I/O -

Streams are implemented in java with "Decorator Pattern"

Input/OutputStreams - these are byte level streams and they read/write bytes i.e., raw data and not characters

Reader/Writer - these are text/character based streams and they read/write ASCII characters

Annotations -

The basic idea behind annotation is Metadata i.e., data about the code or data about the data

Annotations are described in the code or structures of the code and it doesn't directly affect the code because these are not executed along with the code

Annotations in java are basically a data holding class i.e., a collection of data that we use to describe some code in our java class

Annotations can be applied to classes, fields, methods and even to single statement level e.g., @SuppressWarnings(value="deprecation")

Annotations are defined via the @interface annotation before the class name

Annotations are mostly used by compilers that helps it to compile the code correctly or to treat certain code that is being compiled in a certain way.

Annotations help in making the code safe by enforcing contracts before the code is run e.g., @Override - gives the compiler a hint, so that it will cause an error and not compile the code if it is not actually overriding something.

Certain tools use annotations to generate code (ORM), documentation (javadoc) and to find the methods intended to be tests in a testing frameworks.

Annotations are also used in the actual runtime, with Serialization, IoC and ORM.

Reflection -

Reflection is just some code that can examine itself and even change itself at runtime

Reflection is not efficient!

Threads -

A thread is a single sequential flow of control and it takes an independent path of execution within a program

Threads are basically used to do background processing and parallel processing when the order of execution doesn't matter

Threads utilize idle CPU time

Threads share memory/data but Processes don't

ALL THE BEST