Core Java

What is the main method in Java? main method is the entry point of any java program. You can only change the name of String array argument. Public static void main(String[] arg)

**Control statements control** the order of execution in a program, based on data values and conditional logic. There are three main categories

of **control** flow **statements**; Selection **statements**: if, if-else and switch. Transfer **statements**: break, continue, return, try-catch-finally and assert.

What kinds of variables can I use with switch statements? **byte**, **short**, **char**, and **int** primitive data types. It also works with enumerated types, the **String** class, and wrapper

class: **Character**, **Byte**, **Short**, and **Integer**

**Objects** have states and behaviors. An **object** is an instance of a class, has a set of methods.

**State and behavior**: state tells us about the type of that object whereas behavior tells us about the operations that the object can perform.

**A class** describes a set of objects with same behavior, a blueprint for objects.

An **object reference** specifies the location of an object. The null reference refers to no object.

Primitive type variables store values. Object variables store reference.

**Primitive data types**

* **byte** h; 1 byte
* **Boolean b; // T/F**
* **Char c; 2 bytes**
* **Short g; small numbers – 2 bytes**
* **int** a; 4 bytes
* **float f; floating decimal – 4 bytes**
* **double** d; // decimal - 8 bytes
* **long** w; // large number - 8 bytes

**Mutable** object – You can change the states and fields after the object is created. For **examples**: StringBuilder , **java**.util.Date and etc. ... Immutable object – You cannot change anything after the object is created. For **examples**: String, boxed primitive objects like Integer, Long and etc.

A **constructor** is a block of code similar to a method that's called when an instance of an object is created. A **constructor** doesn't have a return type. ... Unlike methods, **constructors** are not considered members of a class. You cannot invoke a constructor.

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.  
Constructor chaining can be done in two ways:

* **Within same class**: It can be done using **this()** keyword for constructors in same class
* **From base class:**by using **super()** keyword to call constructor from the base class.

**Inheritance:** is a mechanism in which one object acquires all the properties and behaviors of a parent object.

**Abstraction**: is a process of hiding the implementation details from the user. Оnly the functionality will be provided to the user. **Abstraction** is achieved using abstract classes and interfaces.

public abstract class MyAbstractClass{ }

public abstract int getLength();

**Interface:**

**public** **interface** Measurable {

**double** getMeasure();

default void talk(“Hello”);

}

Class can’t extend two classes, but implements multiple interface.

**Encapsulation**: is the act of providing a public interface and hiding the implementation details.

* Private instance variables,
* Public getter (accessor methods) & setters(mutator methods)

**Polymorphisms** Ability for objs to behave in various manners

* Method overloading: same name, different signature (same class)
* Method overriding: same name, different implementation (inherited classes)

Covariant types ie. Human h = new Adult();

**Package:** A package is a namespace that organizes a set of related classes and interfaces. You can think of packages as being similar to different folders on your computer. Sub packages have no special connection to parent packages. i.e com.ex.\* will not import com.ex.sub.ClassName

**OOP**: is a **programming** language model organized around **objects** rather than "actions" and data rather than logic.

**Java is not fully OOP because it has primitive types. Java is passed by value.**

JVM: compile java, java virtual machine -> generate .class file, Byte code, Café babe.

JRE: java runtime environment, execute.

JDK: Java development kit

java c helloworld.java compile

java HelloWorld execute

java HelloWorld "these are parameters"

**Wrapper class**

Auto-boxing: Conversion between primitive types and the corresponding wrapper class is automatic.

if you assign a double value to a Double variable, the number is automatically put into a box.

Double wrapper = 29.95;

Conversely, wrapper values are automatically unboxed to primitive types.

double x = wrapper;

**Casting:** to convert a value to a different type.

double balance = 12.75;

int dollar = (int) (balance + tax); lost faction

dollar = 12;

or int ronded = (int) Math.round(balance);

rounded = 13;

**Strings Class**

java.lang.String

Java.lang is only package in Java that we can access classes directly without the need of import

Special objects in Java

Not the same thing as Character[]

String literal =" " can filled w >= char

String are immutable, cannot change its contents after created.

String can be initialized in 2 ways

String st="hello", String str2 = new String("hi");

gets stored in the String pool: String st="hello",

gets stored as object in the heap: String str2 = new String("hi");

**String builder**-mutable Strings: fast, not synchronized

**String buffered**: thread safe, mutable Strings, slower but synchronized.

**Memory in Java: Stack memory, heap space**

Java heap space is used by Java runtime to allocate memory to objs & JRE classes. Garbage collection is used to free memory from any unreferenced objs. Contains the String pool.

Java stack memory: used for execution of a thread, or a single chronological process. It also contains local (method specific) values & referenced to objs in the heap being referred to in that method. Stack memory is always referenced in LIFO order.

What is the **String pool**? An area in heap memory where Java store literal String values.

**Access modifiers/levels**

Public String aMethod(){}

* The public access modifier is accessible everywhere.
* The protected modifier is accessible within the same package and from subclasses.
* A private access modifier is accessible only within the class it is declared in.
* A modifier can also not be used which is known as “default.” It can only be accessed within the same package.

An object’s instance variables store the data required for executing its methods. An object stores its data in instance variables. An instance of a class is an object of the class.

Public void sum(int…num) {

Num l = num.length;

}

Sum();

Sum(1,2);

**Variable scopes:** scope of a variable is the part of the program in which it is visible.

. Instance / object

. static / class

. local / method

. block/loop scope

There are 4 scopes

1. static/class – belongs to the class, not to any object of the class, and accessible from outside of the class without an instance.

Regarding static variables, these values are shared throughout any instances of the class.

2. object/instance - the particular instance of the objects

\* state and behavior

\* Class c = new Class();

\* c.x; or c.method();

3. method/local - parameters for method and any variable declare within the method.

4. block/loop - any variable defined in curly braces or in the () before loop

Static block which can be used for static initializations of a class. This code inside static block is executed only once: the first time you make an object of that class or the first time you access a static member of that class (even if you never make an object of that class).

**Static methods**: methods that are not invoked on objects. Math.pow, primitive types are not objects.

**Instance methods**: methods that are invoked on objects.

**Errors**: outofmemory, stackoverflow

**Checked exception** is an exception that is checked (notified) by the compiler at compilation-time

**Unchecked exception** is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions: IOException, IndexOutofBound, InputMishaspexception**

*Throws(after method)* is used to postpone the handling of a checked exception and *throw* is used to invoke an exception explicitly.

**Unit Testing**

Most granular form of software testing.

Evaluating smallest executable pieces of code (methods)

Performed by programmer – not team of external testers, should always think “TDD” – Test Driven Development.

Junit – popular Java Unit Testing framework, must use external jar for org.junit packages.

**Annotations:**

**@Test – indicates that the following method is a unit test**

**@Before @ After – run before/after each @Test method**

**@BeforeClass/@AfterClass - run before/after all @Test methods in the class are executed.**

**@Assert methods – methods used to define expected behavior & indicate success/failure.**

**@Runwith, @SuiteCases : Test Suit**

**@Override**

**@Functional interface**

Interfaces

* An abstract type that is used to define behavior that subclasses must implement “Contract”
* Classes may implement multiple interfaces -> methods are implicitly public & abstract.
* Since Java 1.8, methods CAN be implemented in interfaces, they must be labeled with the default keyword.
  + All variables are implicitly public static final.
  + “Abstract” keyword – apply to a method to require it to be overridden in subclass.
  + Related functional interfaces, marker interfaces, multiple inheritance.

**Abstract Classes**

* May have both concrete & abstract methods.
* Cannot be instantiated.

A *functional interface* is an interface that contains only a single abstract (unimplemented) method

The interfaces with no defined methods act like markers. They just tell the compiler that the objects of the classes implementing the interfaces with no defined methods need to be treated differently. Marker interfaces are also known as “tag” interfaces.

Final keyword:

-Final class – cannot be extended.

-Final method – cannot be overridden. CAN be overloaded.

-Final variables – cannot be reassigned constant values. i.e final static int MAX\_VALUE = 100;

- **Thread** – a thread is the path followed when executing a program. All Java programs have at least one thread known as the main thread, which is created by the JVM at the program’s start, when the main() method is invoked.

-Multithreading – multiple flows of threads of execution in program execution.

-Each thread gets its own stack.

-Create a separate thread by extending the Thread class or by implementing the Runnable interface overriding the run() method of the Runnable interface, then calling the start() method of the Thread class.

Thread class -> worker

Start();

Runnable interface -> job

Run();

public class MyThread extends Thread {

public void run(){

System.out.println("MyThread running");

}

}

Main() {

MyThread a = new MyThread();

a.start();

}

The third way to implement the Runnable interface is by creating a [**Java Lambda**](http://tutorials.jenkov.com/java/lambda-expressions.html) implementation of the Runnable interface. This is possible because the Runnable interface only has a single unimplemented method, and is therefore practically a [**functional Java interface**](http://tutorials.jenkov.com/java-functional-programming/functional-interfaces.html).

Runnable runnable =

() -> { System.out.println("Lambda Runnable running"); };

JVM Thread scheduler – as a developer, you don’t have complete control over thread execution, But you can influence via thread priority (1-10).

Thread States: New, RUNNABLE, BLOCKED, WAITING, TIMED-WAITING, & TERMINATED.

Types of threads:

* User-defined: main + any other created threads.
* Daemon – background processes. i.e Garbage collector.

Related: synchronized keyword, deadlock, starvation, produce, consumer problem.

**Deadlock** describes a situation where two or more threads are blocked forever, waiting for each other ... Use synchronized to prevent deadlock.

Anomous class:

Thread thread = new Thread(){

public void run(){

System.out.println("Thread Running");

}

}

thread.start();

**Serialization** is a mechanism of converting the state of an object into a byte stream. To make a Java object serializable we implement the **java.io.Serializable** interface.

**Advantages of Serialization**

1. To save/persist state of an object.  
2. To travel an object across a network.

Garbage collector reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. Any unreferenced object is a garbage and will be collected. Before an object is garbage collected, the

runtime system calls its finalize() method.

Transient variables cannot be serialized. During serialization process,

transient variable states will not be serialized. State of the value will

be always defaulted after deserialization.

**Design patterns** are established general solutions to commonly occurring software development problems.

The singleton design pattern is a pattern which creates an object in memory only once throughout the lifetime of the application to be shared across multiple classes.

\* To make a singleton you must make a private static variable of the class's type, make a private constructor, and make a public static getInstance() method, which will return the single instance of the class created.

\* There are two types of singletons

\* Lazy Singleton

\* Eager Singleton

The factory design pattern is a pattern which creates objects where the precise type may not be known until runtime (create a reference to a parent class or interface but instantiate with the particular implementing class based on some sort of user input)

What make String class immutable? caching, security, synchronization, and performance

* **== compares two objects by reference to location in memory. equals () comparing the values of the two objects.**

Maven: Java build tool, manage dependencies, build our project.

POM: Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

**Reflection** is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime. Through **reflection** we can invoke methods at runtime irrespective of the access specifier used with them.

Collection: a group of individual objects represented as a single unit. We use them to store and organize our groups of objects. Collection is interface, collections is utility class which has a lot of static methods.

Collections.sort;

Hashsets are sets without duplicates, there is no guarantee about the order of elements in a hashset.

Set<String> set = new HashSet<>();

TreeSet sortedSet = new TreeSet<Integer>(set);

Iterator<String> iter = set.iterator();

Methods: add, contain, remove, size, clear, isempty, iterator().

Maps map keys to values

Methods: containKey, containValue, clear, equal, get, put, remove, size, keyset.

Map<Integer, String> map = new HashMap<>();

Map.put(1, “hello”);

Get, remove

Set<String> keyset = map.keySet(); // Retrieve a key

For(String key : keyset) {

Value = map.get(key)

Queue: Queue<String> q = new ArrayDeque<>();

q.add

Iterator<String> I = q.iterator();

While(i.hasNext())

Print(i)

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |