When Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.

A class consist of Local variable, instance variables and class variables.

Class variables or static variables are variables declared with in a class, outside any method, with the static keyword.

Singleton class control object creation, limiting the number to one but allowing the flexibility to create more objects if the situation changes.

package com.myjava.constructors;

public class MySingleTon {

    private static MySingleTon myObj;

    /\*\*

     \* Create private constructor

     \*/

    private MySingleTon(){

    }

    /\*\*

     \* Create a static method to get instance.

     \*/

    public static MySingleTon getInstance(){

        if(myObj == null){

            myObj = new MySingleTon();

        }

        return myObj;

    }

    public void getSomeThing(){

        // do something here

        System.out.println("I am here....");

    }

    public static void main(String a[]){

        MySingleTon st = MySingleTon.getInstance();

        st.getSomeThing();

    }

}

A member has package or default accessibility when no accessibility modifier is specified.

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

What do you mean by synchronized Non Access Modifier? Java provides these modifiers for providing functionalities other than Access Modifiers, synchronized used to indicate that a method can be accessed by only one thread at a time.

Postfix operators i.e () [] . is at the highest precedence. Ex: funHelp()

Why is String class considered immutable? The String class is immutable, so that once it is created a String object cannot be changed. Since String is immutable it can safely be shared between many threads, which is considered very important for multithreaded programming.

What is the difference between StringBuffer and StringBuilder class? Use StringBuilder whenever possible because it is faster than StringBuffer. But, if thread safety is necessary then use StringBuffer objects.

What is finalize() method? It is possible to define a method that will be called just before an object's final destruction by the garbage collector. This method is called finalize( ), and it can be used to ensure that an object terminates cleanly.

GarbageCollection:<https://www.dynatrace.com/resources/ebooks/javabook/how-garbage-collection-works/>

Checked exception – Compile time exception

Exception class has two main subclasses : IOException class and RuntimeException Class.

When throws keyword is used? If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature.

When throw keyword is used? An exception can be thrown, either a newly instantiated one or an exception that you just caught, by using throw keyword.

While creating your own exception −

* All exceptions must be a child of Throwable.
* If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
* You want to write a runtime exception, you need to extend the RuntimeException class.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Interface cannot be instantiated

An interface does not contain any constructors.

All of the methods in an interface are abstract.

Thread can be created by: implementing Runnable interface, extending the Thread class.

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. An applet extends java.applet.Applet class.

Set Interface: It is a collection of element which cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

TreeSet: It is a Set implemented when we want elements in a sorted order.

Comparable Interface: It is used to sort collections and arrays of objects using the collections.sort() and java.utils. The objects of the class implementing the Comparable interface can be ordered.

Without throws, Checked exception cannot be handled where as checked exception can be propagated with throws.

static − it allows main() to be called without instantiating a particular instance of a class.

Java Runtime Environment is an implementation of the Java Virtual Machine which executes Java programs. It provides the minimum requirements for executing a Java application;

JAR files is Java Archive fles and it aggregates many files into one. It holds Java classes in a library.

WAR is Web Archive File and used to store XML, java classes, and JavaServer pages. which is used to distribute a collection of JavaServer Pages, Java Servlets, Java classes, XML files, static Web pages etc.

Object based programming languages follow all the features of OOPs except Inheritance. JavaScript is an example of object based programming languages.

A constructor cannot be made final.

What is static block? It is used to initialize the static data member, It is executed before main method at the time of classloading.

Holding the reference of the other class within some other class is known as composition.

Composition over Inheritance: <https://www.journaldev.com/1775/multiple-inheritance-in-java>

Final classes are created so the methods implemented by that class cannot be overridden. It can’t be inherited.

A thread can enter the waiting state by invoking its sleep() method, by blocking on IO, by unsuccessfully attempting to acquire an object's lock, or by invoking an object's wait() method. It can also enter the waiting state by invoking its (deprecated) suspend() method.

How does multi-threading take place on a computer with a single CPU? The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching between executing tasks, it creates the impression that tasks execute sequentially.

What invokes a thread's run() method? After a thread is started, via its start() method of the Thread class, the JVM invokes the thread's run() method when the thread is initially executed.

Does it matter in what order catch statements for FileNotFoundException and IOException are written? Yes, it does. The FileNotFoundException is inherited from the IOException. Exception's subclasses have to be caught first.

What is the difference between yielding and sleeping? When a task invokes its yield() method, it returns to the ready state. When a task invokes its sleep() method, it returns to the waiting state.

The Vector class provides the capability to implement a growable array of objects. Vector proves to be very useful if you don't know the size of the array in advance, or you just need one that can change sizes over the lifetime of a program.

Wrapper classes are classes that allow primitive types to be accessed as objects. Example: Integer, Character, Double, Boolean etc.

File class is used to create objects that provide access to the files and directories of a local file system.

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented.

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

Can you write a Java class that could be used both as an applet as well as an application? Yes, just add a main() method to the applet.

When is the ArrayStoreException thrown? When copying elements between different arrays, if the source or destination arguments are not arrays or their types are not compatible, an ArrayStoreException will be thrown.

What's the difference between the methods sleep() and wait()? The code sleep(2000); puts thread aside for exactly two seconds. The code wait(2000), causes a wait of up to two second. A thread could stop waiting earlier if it receives the notify() or notifyAll() call. The method wait() is defined in the class Object and the method sleep() is defined in the class Thread.

A transient variable is a variable that may not be serialized during Serialization and which is initialized by its default value during de-serialization,

Does garbage collection guarantee that a program will not run out of memory? Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection.

The immediate superclass of the Applet class? Panel is the immediate superclass. A panel provides space in which an application can attach any other component, including other panels.

The = operator is right associative.

What is the purpose of the System class? The purpose of the System class is to provide access to system resources.

What class of exceptions are generated by the Java run-time system? The Java runtime system generates RuntimeException and Error exceptions.

Under what conditions is an object's finalize() method invoked by the garbage collector? The garbage collector invokes an object's finalize() method when it detects that the object has become unreachable.

A class loader is an object that is responsible for loading classes. The class ClassLoader is an abstract class.

An abstract class can have instance methods that implement a default behavior. An Interface can only declare constants and instance methods, but cannot implement default behavior and all methods are implicitly abstract. An interface has all public members and no implementation.

No, a top level class can not be private or protected. It can have either "public" or no modifier.

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. Exceptions are conditions that occur because of bad input etc. e.g. FileNotFoundException will be thrown if the specified file does not exist.

A thread is in the ready state as initial state after it has been created and started.

Synchronized methods are methods that are used to control access to an object. A synchronized statement can only be executed after a thread has acquired the lock for the object or class referenced in the synchronized statement.

What is Dynamic Binding(late binding)? Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run-time.

constructor cannot be inherited.

ArrayList can grow dynamically and provides more powerful insertion and search mechanisms than arrays.

What is a Values Collection View ? It is a collection returned by the values() method of the Map Interface, It contains all the objects present as values in the map.

What is the difference between the >> and >>> operators? >> Arithmetic right shift (Sign not neglected and will remain same even after any number of shifts). >>> Logical shift (Simply shift and append 0).

Java does not allow Default Arguments.

Break statement can be used as labels in Java? Yes, an example can be break one;

If a class is declared within a class and specify the static modifier, the compiler treats the class just like any other top-level class. Nested top-level class is an Inner class.

If System.exit (0); is written at the end of the try block, will the finally block still execute? No in this case the finally block will not execute because when you say System.exit (0); the control immediately goes out of the program, and thus finally never executes.

When a class is defined within a scope of another class, then it becomes inner class. If the access modifier of the inner class is static, then it becomes nested class.

Constructor Chaining: A child object constructor always first needs to construct its parent. In Java it is done via an implicit call to the no-args constructor as the first statement.

**Java 8 Interface changes**

1. Default Method

public interface Interface1 {

void method1(String str);

default void log(String str){

System.out.println("I1 logging::"+str);

}

}

Java interface default methods are also referred to as Defender Methods or Virtual extension methods.

2. Static Method

Java interface static method is similar to default method except that we can’t override them in the implementation classes.

Java interface static method is part of interface, we can’t use it for implementation class objects.

Java interface static method helps us in providing security by not allowing implementation classes to override them.

We can’t define interface static method for Object class methods, we will get compiler error as “This static method cannot hide the instance method from Object”. This is because it’s not allowed in java, since Object is the base class for all the classes and we can’t have one class level static method and another instance method with same signature.

**Java 8 Functional Interfaces**

An interface with **exactly one abstract method** is called Functional Interface. **@FunctionalInterface annotation** is added so that we can mark an interface as functional interface.

It is not mandatory to use it, but it’s best practice to use it with functional interfaces to avoid addition of extra methods accidentally. If the interface is annotated with @FunctionalInterface annotation and we try to have more than one abstract method, it throws compiler error.

Runnable r = new Runnable(){

@Override

public void run() {

System.out.println("My Runnable");

}};

The major benefit of java 8 functional interfaces is that we can use lambda expressions to instantiate them and avoid using bulky anonymous class implementation.

java.lang.Runnable is a great example of functional interface with single abstract method run().

**Lambda Expression** are the way through which we can visualize functional programming in the java object oriented world. Since there is **only one abstract function in the functional interfaces**, there is no confusion in applying the lambda expression to the method. Lambda Expressions syntax is **(argument) -> (body)**.

Runnable r1 = () -> System.out.println("My Runnable");

Runnable is a functional interface, that’s why we can use lambda expression to create it’s instance.

Java Stream vs Collection

A collection is an in-memory data structure to hold values and before we start using collection, all the values should have been populated. Whereas a java Stream is a data structure that is computed on-demand.

Java Stream doesn’t store data, it operates on the source data structure (collection and array) and produce pipelined data that we can use and perform specific operations. Such as we can create a stream from the list and filter it based on a condition.

Java Streams are consumable, so there is no way to create a reference to stream for future usage. Since the data is on-demand, it’s not possible to reuse the same stream multiple times.

**Spring**

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. DI helps you decouple your application objects from each other, while AOP helps you decouple cross-cutting concerns from the objects that they affect.

The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code.



2 types of Spring Container

1. Spring BeanFactory Container (usually preferred where the resources are limited like mobile devices or applet-based applications.)
2. Spring ApplicationContext Container

The Application Context is Spring's advanced container. Similar to BeanFactory, it can load bean definitions, wire beans together, and dispense beans upon request. Additionally, it adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. Commonly used ApplicationContext Implementations

1. FileSystemXmlApplicationContext
2. ClassPathXmlApplicationContext
3. WebXmlApplicationContext

The three important methods to provide configuration metadata to the Spring Container

1. XML based configuration file
2. Annotation-based configuration
3. Java-based configuration

Scope Attribute of beans

1. Singleton (default)

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

1. Prototype

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made. As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

1. request
2. session
3. global-session

Among these, last 3 are available only if we use a web aware Spring ApplicationContext.

You register a shutdown hook registerShutdownHook() method that is declared on the AbstractApplicationContext class with the JVM. Doing so ensures a graceful shutdown and calls the relevant destroy methods on your singleton beans so that all resources are released.

The BeanPostProcessor interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, etc. You can also implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean by plugging in one or more BeanPostProcessor implementations.

You can configure multiple BeanPostProcessor interfaces and you can control the order in which these BeanPostProcessor interfaces execute by setting the order property provided the BeanPostProcessor implements the Ordered interface.

The BeanPostProcessors operate on bean (or object) instances, which means that the Spring IoC container instantiates a bean instance and then BeanPostProcessor interfaces do their work.

An ApplicationContext automatically detects any beans that are defined with the implementation of the BeanPostProcessor interface and registers these beans as postprocessors, to be then called appropriately by the container upon bean creation.

This is the bean definition template which can act parent class for other beans. This alone can’t be a instantiated as its incomplete bean. Its abstract.

<bean id = "beanTeamplate" abstract = "true">

<property name = "message1" value = "Hello World!"/>

<property name = "message2" value = "Hello Second World!"/>

<property name = "message3" value = "Namaste India!"/>

</bean>

**Dependency Injection**: Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

1. Constructor based DI

* It is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class.
* In case you are passing a reference to an object, you need to use ref attribute of <constructor-arg> tag and if you are passing a value directly then you should use value attribute.

1. Setter based DI

* It is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.
* In case you are passing a reference to an object, you need to use ref attribute of <property> tag and if you are passing a value directly then you should use value attribute.

**Inner Beans**

A <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean.

**Beans Autowiring:**

The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application. By default, no autowiring will be done.

Modes:-

1. Default is no autowiring
2. **byName**: In the variable name in the class, if it matches with any of the available beans, then that bean gets autowired to that variable of the class which in turn is be a bean.
3. **byType**: The variable’s(object) class(or type), if it matches with any of the available beans, then that bean gets autoiwired to that variable.
4. **constructor**: In the class’s constructor, if any of the argument matches the type of any of the available beans, then that bean gets autowired to that particular constructor argument.
5. **autodetect:** Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType.

You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring.

You cannot autowire so-called simple properties such as primitives, Strings, and Classes.

Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring.

**Annotation based DI:** Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

Annotation wiring is not turned on in the Spring container by default. Below line has to be used in configuration file.

<context:annotation-config/>

@Required : The @Required annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time.

@Autowired:

* @Autowired on Setter methods: You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file. When Spring finds an @Autowired annotation used with setter methods, it tries to perform byType autowiring on the method.
* @Autowired on Properties: You can use @Autowired annotation on properties to get rid of the setter methods. When you pass the values of autowired properties using <property>, Spring will automatically assign those properties with the passed values or references.
* @Autowired on Constructors: A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used while configuring the bean in XML file. Let us check the following example.
* @Autowired with (required=false) option: By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using the (required=false) option with @Autowired.

@Qualifier: There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired.

Ex: @Qualifier("student1")

@JSR-250 Annotations:

* @PostConstruct and @PreDestroy Annotations: To define the setup and teardown for a bean, we simply declare the <bean> with init-method and/or destroy-method parameters. The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation. Similarly, the destroy-method specifies a method that is called just before a bean is removed from the container. You can use @PostConstruct annotation as an alternate of initialization callback and @PreDestroy annotation as an alternate of destruction callback.
* @Resource Annotation: You can use @Resource annotation on fields or setter methods and it works the same as in Java EE 5. The @Resource annotation takes a 'name' attribute which will be interpreted as the bean name to be injected.

**Java Based Configuration instead of XML**

@Configuration & @Bean Annotations:

Annotating a class with the @Configuration indicates that the class can be used by the Spring IoC container as a source of bean definitions.

The @Bean annotation tells Spring that a method annotated with @Bean will return an object that should be registered as a bean in the Spring application context. The method name is annotated with @Bean works as bean ID and it creates and returns the actual bean.

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class HelloWorldConfig {

@Bean

public HelloWorld helloWorld(){

return new HelloWorld();

}

}

The above code will be equivalent to the following XML configuration –

<beans>

<bean id = "helloWorld" class = "com.tutorialspoint.HelloWorld" />

</beans>

Your configuration class can have a declaration for more than one @Bean. You can load various configuration classes as follows –

public static void main(String[] args) {

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();

ctx.register(AppConfig.class, OtherConfig.class);

ctx.register(AdditionalConfig.class);

ctx.refresh();

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

Injecting Bean Dependencies:

When @Beans have dependencies on one another, expressing that the dependency is as simple as having one bean method calling another as follows

package com.tutorialspoint;

import org.springframework.context.annotation.\*;

@Configuration

public class AppConfig {

@Bean

public Foo foo() {

return new Foo(bar());

}

@Bean

public Bar bar() {

return new Bar();

}

}

Here, the foo bean receives a reference to bar via the constructor injection.

The @Import Annotation: The @Import annotation allows for loading @Bean definitions from another configuration class.

Consider a ConfigA class as follows –

@Configuration

public class ConfigA {

@Bean

public A a() {

return new A();

}

}

You can import above Bean declaration in another Bean Declaration as follows

@Configuration

@Import(ConfigA.class)

public class ConfigB {

@Bean

public B a() {

return new A();

}

}

Now, rather than needing to specify both ConfigA.class and ConfigB.class when instantiating the context, only ConfigB needs to be supplied as follows –

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(ConfigB.class);

// now both beans A and B will be available...

A a = ctx.getBean(A.class);

B b = ctx.getBean(B.class);

}

Lifecycle Callbacks: The @Bean annotation supports specifying arbitrary initialization and destruction callback methods, much like Spring XML's init-method and destroy-method attributes on the bean element –

public class Foo {

public void init() {

// initialization logic

}

public void cleanup() {

// destruction logic

}

}

@Configuration

public class AppConfig {

@Bean(initMethod = "init", destroyMethod = "cleanup" )

public Foo foo() {

return new Foo();

}

}

Specifying Bean Scope:

The default scope is singleton, but you can override this with the @Scope annotation as follows –

@Configuration

public class AppConfig {

@Bean

@Scope("prototype")

public Foo foo() {

return new Foo();

}

}

Event handling in Spring:

The ApplicationContext publishes certain types of events when loading the beans. For example, a ContextStartedEvent is published when the context is started and ContextStoppedEvent is published when the context is stopped.

Event handling in the ApplicationContext is provided through the ApplicationEvent class and ApplicationListener interface. Hence, if a bean implements the ApplicationListener, then every time an ApplicationEvent gets published to the ApplicationContext, that bean is notified.

Spring provides the following standard events –

1. ContextRefreshedEvent
2. ContextStartedEvent
3. ContextStoppedEvent
4. ContextClosedEvent
5. RequestHandledEvent

Spring's event handling is single-threaded so if an event is published, until and unless all the receivers get the message, the processes are blocked and the flow will not continue. Hence, care should be taken when designing your application if the event handling is to be used.

AOP with Spring:

Aspect-Oriented Programming entails breaking down program logic into distinct parts called so-called concerns. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects like logging, auditing, declarative transactions, security, caching, etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other and AOP helps you decouple cross-cutting concerns from the objects that they affect.

Spring AOP module provides interceptors to intercept an application. For example, when a method is executed, you can add extra functionality before or after the method execution.

(Least Accessible) private < default < protected < public (Most Accessible)

Easy way to remember is that default access is more restricted than protected and protected members are accessible in subclasses.

16. What is final keyword?

final keyword is used with Class to make sure no other class can extend it, for example String class is final and we can’t extend it. We can use final keyword with methods to make sure child classes can’t override it. final keyword can be used with variables to make sure that it can be assigned only once. However the state of the variable can be changed, for example we can assign a final variable to an object only once but the object variables can change later on. Java interface variables are by default final and static.

Usually static methods are utility methods that we want to expose to be used by other classes without the need of creating an instance. For example Collections class.

Static nested classes can access only static members of the outer class.

OuterClass.StaticNestedClass nestedObject =

new OuterClass.StaticNestedClass();

Java inner class is associated with the object of the class and they can access all the variables and methods of the outer class. Since inner classes are associated with instance, we can’t have any static variables in them. Object of java inner class are part of the outer class object and to create an instance of inner class, we first need to create instance of outer class.

OuterClass outerObject = new OuterClass();

OuterClass.InnerClass innerObject = outerObject.new InnerClass();

Java Abstract class can implement interfaces without even providing the implementation of interface methods.

Spring Integration module is built upon Pipes and Filter pattern.

Master-Slave: Databases like MySQL could be configured in a Master- Slave setup. The master node could be read-write node whereas the slave nodes could provide read only operations.

String pool is also example of **Flyweight design pattern**

If more than one member method is both accessible and applicable to a method invocation … The Java programming language uses the rule that the most specific method is chosen.

We are not allowed to assign MyClass<String> variable to MyClass<Object> variable because they are not related, in fact MyClass<T> parent is Object.

Iterators in collection classes implement **Iterator Design Pattern**.

Set interface doesn’t allow random-access to an element in the Collection. You can use iterator or foreach loop to traverse the elements of a Set.

Collections.sort() method that takes Comparator argument follows **Strategy Pattern**.

ArrayList class is roughly equivalent to Vector, except that it is unsynchronized.

HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits null.