Important Sites:

1. <http://websystique.com/spring/spring4-hibernate4-mysql-maven-integration-example-using-annotations/>

Preparation for Interview:

* Introduction
* What is your work in your company?

**Interview Question For Core java**

**Q1. Diffrence between JDK, JVM,JRE?**

* **JVM**  is an acronym for java virtual machine,JVM provide runtime environment in which byte code can be run. JVM are available for many hardware and s/w .It is a specification.It means JVM is platform dependent.
* **JRE** stands for Java Runtime environment . It is Implementation of JVM.
* **JDK** stands for java development kit .It is physically Exist. It Contains JRE and development Kit

**JVM performs following operation:**

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**JVM provides definitions for the:**

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.



**Q2. How many types of memory area are allocated by JVM?**

* Heap(runtime data area in which object are allocated).
* Stack(Java stack stores frames .It holds local variable and partial results and part in method invocation and return).
* Class (method) Area (its stores all per classes structure such as runtime constant pool, method data and code for methods).
* Native Method Area (Native methods used in the application).
* Program Counter Register (It contains the address of JVM instruction currently being exec).

**Note: Class loader is a subsystem of JVM used to load class.**

**Q-3: Difference b/w Final,finally, finalize?**

* **Final**- It is a modifier applicable for classes ,methods and variables. If a class declare as a final then we can’t extend i.e we can’t create child class for that class.

If a method declare as a final we cant override that method.

If a variable define as a final then it will become constant it mean we can’t perform re assignment.

* **Finally:**  it is a block always associated with try catch to maintain clean up code

try {

// Risky code

} catch(){

// Handling

}

finally{

// Clean up code here

}

* **Finalize() :** it is a method present in object class. garbage collector call finalize() just before destroy an object to perform clean up activity.

**Note: Finally meant for cleanup activities related to try block. Whereas finalize meant for cleanup activities related to object.**

**Q: what are the various modifier in java?**

* public
* Private
* Protected
* Default
* Final
* Abstract
* static
* synchronized
* native,
* strictfy
* transient
* volatile

**Question: Dynamic & static binding?**

* **Dynamic binding:** When type of the object is determined at run-time, it is known as dynamic binding.
* **static binding:** When type of the object is determined at compiled time(by the compiler), it is known as static binding.

If there is any private, final or static method in a class, there is static binding

**Java instanceOf:**

The java instanceOf operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

**class** Simple1{

**public** **static** **void** main(String args[]){

 Simple1 s=**new** Simple1();

 System.out.println(s **instanceof** Simple1);//true

 }

}

**Java RMI(Remote Method Invocation)**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

**Understanding stub and skeleton**

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

**stub**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

* It initiates a connection with remote Virtual Machine (JVM),
* It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
* It waits for the result
* It reads (unmarshals) the return value or exception, and
* It finally, returns the value to the caller.

**skeleton**

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

* It reads the parameter for the remote method
* It invokes the method on the actual remote object, and
* It writes and transmits (marshals) the result to the caller.



**JAVA Inner Classes**

**Java inner class** or nested class is a class which is declared inside the class or interface.

**Advantages:**

* We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.
* Nested classes represent a special type of relationship that, it can access all the members of outer class including private data members and methods.
* Code Optimization. It requires less code to write.

**Q. Difference between nested class and inner class in Java**

-Inner classes is part of nested classes. And not-static nested class is called inner classes.

**Types of Nested classes**

1. Non-static nested class (inner class).
   1. Member inner class
   2. Anonymous inner class
   3. Local inner class
2. Static nested class.

[Member Inner Class](https://www.javatpoint.com/member-inner-class): A class created within class and outside method.

[Anonymous Inner Class](https://www.javatpoint.com/anonymous-inner-class): A class created for implementing interface or extending class. Its name is decided by the java compiler.

[Local Inner Class](https://www.javatpoint.com/local-inner-class): A class created within method.

[Static Nested Class](https://www.javatpoint.com/static-nested-class): A static class created within class.

[Anonymous Inner Class](https://www.javatpoint.com/anonymous-inner-class): A class created for implementing interface or extending class. Its name is decided by the java compiler.

[Nested Interface](https://www.javatpoint.com/nested-interface): An interface created within class or interface.

**===============================================================================**

**Java Reflection**

**Java Reflection** is a *process of examining or modifying the run time behavior of a class at run time*.

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection.

The Reflection API is mainly used in:

* IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
* Debugger
* Test Tools etc.

**java.lang.Class class:** The java.lang.Class class performs mainly two tasks:

* provides methods to get the metadata of a class at run time.
* provides methods to examine and change the run time behavior of a class.

There are 3 ways to get the instance of Class class. They are as follows:

* forName() method of Class class
* getClass() method of Object class
* the .class syntax

|  |  |
| --- | --- |
| **Method** | Description |
| **1) public String getName()** | returns the class name |
| **2) public static Class forName(String className)throws ClassNotFoundException** | loads the class and returns the reference of Class class. |
| **3) public Object newInstance()throws InstantiationException,IllegalAccessException** | creates new instance. |
| **4) public boolean isInterface()** | checks if it is interface. |
| **5) public boolean isArray()** | checks if it is array. |
| **6) public boolean isPrimitive()** | checks if it is primitive. |
| **7) public Class getSuperclass()** | returns the superclass class reference. |
| **8) public Field[] getDeclaredFields()throws SecurityException** | returns the total number of fields of this class. |
| **9) public Method[] getDeclaredMethods()throws SecurityException** | returns the total number of methods of this class. |
| **10) public Constructor[] getDeclaredConstructors()throws SecurityException** | returns the total number of constructors of this class. |
| **11) public Method getDeclaredMethod(String name,Class[] parameterTypes)throws NoSuchMethodException,SecurityException** | returns the method class instance. |

### Determining the class object

Following methods of Class class is used to determine the class object:

|  |
| --- |
| **1) public boolean isInterface():** determines if the specified Class object represents an interface type. |
| **2) public boolean isArray():** determines if this Class object represents an array class. |
| **3) public boolean isPrimitive():** determines if the specified Class object represents a primitive type. |

**Object Cloning in Java**

The **object cloning** is a way to create exact copy of an object. The clone() method of Object class is used to clone an object.

The **java.lang.Cloneable interface** must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates **CloneNotSupportedException**.

The **clone() method** is defined in the Object class. Syntax of the clone() method is as follows:

1. **protected** Object clone() **throws** CloneNotSupportedException

### Why use clone() method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the new keyword, it will take a lot of processing time to be performed that is why we use object cloning.

class Student18 implements Cloneable{

int rollno;

String name;

Student18(int rollno,String name){

this.rollno=rollno;

this.name=name;

}

@Override

public Object clone()throws CloneNotSupportedException{

return super.clone();

}

public static void main(String args[]){

try{  Student18 s1=new Student18(101,"amit");

Student18 s2=(Student18)s1.clone();

System.out.println(s1.rollno+" "+s1.name);

System.out.println(s2.rollno+" "+s2.name);

}catch(CloneNotSupportedException c){}  }

}

**Difference between abstract class and interface**

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Example of abstract class and interface in Java

//Creating interface that has 4 methods

**interface** A{

**void** a();//bydefault, public and abstract

**void** b();

**void** c();

**void** d();

}

//Creating abstract class that provides the implementation of one method of A interface

**abstract** **class** B **implements** A{

**public** **void** c(){System.out.println("I am C");}

}

  //Creating subclass of abstract class, now we need to provide the implementation of rest of the methods

**class** M **extends** B{

**public** **void** a(){System.out.println("I am a");}

**public** **void** b(){System.out.println("I am b");}

**public** **void** d(){System.out.println("I am d");}

}

//Creating a test class that calls the methods of A interface

**class** Test5{

**public** **static** **void** main(String args[]){

A a=**new** M();

a.a();

a.b();

a.c();

a.d();

}}

**Java Packages**

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

**Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

# Java Regex

The **Java Regex** or Regular Expression is an API to define a pattern for searching or manipulating strings.

It is widely used to define the constraint on strings such as password and email validation. After learning Java regex tutorial, you will be able to test your regular expressions by the Java Regex Tester Tool.

Java Regex API provides 1 interface and 3 classes in **java.util.regex** package.

#### java.util.regex package

The Matcher and Pattern classes provide the facility of Java regular expression. The java.util.regex package provides following classes and interfaces for regular expressions.

1. MatchResult interface
2. Matcher class
3. Pattern class
4. PatternSyntaxException class

## Matcher class

It implements the **MatchResult** interface. It is a regex engine which is used to perform match operations on a character sequence.

## Pattern class

It is the compiled version of a regular expression. It is used to define a pattern for the regex engine.

**=========================================**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**next>>**](https://www.javatpoint.com/interrupting-a-thread)[**<<prev**](https://www.javatpoint.com/deadlock-in-java) Inter-thread communication in Java **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.  Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:   * wait() * notify() * notifyAll()  1) wait() method Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.  The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.   |  |  | | --- | --- | | **Method** | **Description** | | public final void wait()throws InterruptedException | waits until object is notified. | | public final void wait(long timeout)throws InterruptedException | waits for the specified amount of time. |  2) notify() method Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:  public final void notify() 3) notifyAll() method Wakes up all threads that are waiting on this object's monitor. Syntax:  public final void notifyAll() Understanding the process of inter-thread communication inter thread communication in java  The point to point explanation of the above diagram is as follows:   1. Threads enter to acquire lock. 2. Lock is acquired by on thread. 3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits. 4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state). 5. Now thread is available to acquire lock. 6. After completion of the task, thread releases the lock and exits the monitor state of the object.  Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class? It is because they are related to lock and object has a lock.  **Difference between wait and sleep?**   |  |  | | --- | --- | | **wait()** | **sleep()** | | wait() method releases the lock | sleep() method doesn't release the lock. | | is the method of Object class | is the method of Thread class | | is the non-static method | is the static method | |  |  | | should be notified by notify() or notifyAll() methods | after the specified amount of time, sleep is completed. | |

**===============================================================================**

**Java Restful Api**

Google Duplex.

Google IO