Interview questions

1. Class loaders in java

The Java Classloader is a part of the Java Runtime Environment that dynamically loads Java classes into the **Java Virtual Machine**. Usually classes are only loaded on demand

ClassLoader in Java works on three principle: delegation, visibility and uniqueness. Delegation principle forward request of class loading to parent class loader and only loads the class, if parent is not able to find or load class. Visibility principle allows child class loader to see all the classes loaded by parent ClassLoader, but parent class loader can not see classes loaded by child. Uniqueness principle allows to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by parent

.

There are three types of built-in ClassLoader in Java:

1) Bootstrap ClassLoader - It loads JDK internal classes, typically loads rt.jar and other core classes for example java.lang.\* package classes

1. Extension ClassLoader - It loads classes from the JDK extensions directory, usually $JAVA\_HOME/lib/ext directory.
2. Application/System ClassLoader  It loads classes from the current classpath that can be set while invoking a program using -cp or -classpath command line options.
3. public class ClassLoaderTest {
4. public static void main(String[] args) {
5. System.out.println("class loader for HashMap: "
6. + java.util.HashMap.class.getClassLoader());
7. System.out.println("class loader for DNSNameService: "
8. + sun.net.spi.nameservice.dns.DNSNameService.class
9. .getClassLoader());
10. System.out.println("class loader for this class: "
11. + ClassLoaderTest.class.getClassLoader());
12. System.out.println(com.mysql.jdbc.Blob.class.getClassLoader());
13. }
14. }

Output:

class loader for HashMap: null

class loader for DNSNameService: sun.misc.Launcher$ExtClassLoader@7c354093

class loader for this class: sun.misc.Launcher$AppClassLoader@64cbbe37

sun.misc.Launcher$AppClassLoader@64cbbe37

java.util.HashMap ClassLoader is coming as null that reflects Bootstrap ClassLoader whereas DNSNameService ClassLoader is ExtClassLoader. Since the class itself is in CLASSPATH, System ClassLoader loads it.

When we are trying to load HashMap, our System ClassLoader delegates it to the Extension ClassLoader, which in turns delegates it to Bootstrap ClassLoader that found the class and load it in JVM.

The same process is followed for DNSNameService class but Bootstrap ClassLoader is not able to locate it since its in $JAVA\_HOME/lib/ext/dnsns.jar and hence gets loaded by Extensions Class Loader.

Why write a Custom ClassLoader in Java?

Java default ClassLoader can load files from local file system that is good enough for most of the cases. But if you are expecting a class at the runtime or from FTP server or via third party web service at the time of loading the class then you have to extend the existing class loader. For example, AppletViewers load the classes from remote web server.

Refer- <http://javarevisited.blogspot.com/2012/12/how-classloader-works-in-java.html>

<http://www.journaldev.com/349/java-classloader>

1. **Return type of main method? Can we override/overload main method**- Because Java's main method is built to accept only void as return type. If you want to return an int, then use System.exit(3434) to return an int when program exits.

Valid main method signatures –

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

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

public static void main(String... args){}

You can also use certain modifier like final, synchronized and strictfp along with main method in Java.

Refer -http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html

1. Difference between static and init block in java

The static initializer block will be called on loading of the class, and will have no access to instance variables or methods. static blocks are very useful to instantiate static class level variables

The non-static initializer block on the other hand is created on object construction only, will have access to instance variables and methods and will be called at the beginning of the constructor. static block only gets called **once**, no matter how many objects of that type you create.

public class LoadingBlocks {

static{  
System.out.println("Inside static");  
}

{  
System.out.println("Inside init");  
}  
public static void main(String args[]){  
new LoadingBlocks();  
new LoadingBlocks();  
new LoadingBlocks();  
}  
}

Output:

Inside static  
Inside init  
Inside init  
Inside init

1. **Can we have final method in abstract class?**

Yes, it can. But the final method cannot be abstract itself

1. **Can we have abstract method in final class**?- abstract methods are not allowed in final class.  
   If a class has abstract method then the class needs to be declared as abstract and not final.
2. **New features of java 7**

1)**Type inference** : We can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters (<>) as long as the compiler can infer the type arguments from the context.

 For example, the following example does not compile:

List<String> l = new ArrayList<>();

l.add("A");

l.addAll(new ArrayList<>());

In comparison, the following example compiles:

List<? extends String> list2 = new ArrayList<>();

l.addAll(list2);

Prior JDK 7

**Map**<**String**, **List**<**String**>> employeeRecords **=** **new** **HashMap**<**String**, **List**<**String**>>();

**List**<**Integer**> primes **=** **new** **ArrayList**<**Integer**>();

In JDK 7

**Map**<**String**, **List**<**String**>> employeeRecords **=** **new** **HashMap**<>();

**List**<**Integer**> primes **=** **new** **ArrayList**<>();

2)String in switch statement

Prior to JDK 1.7

***if****(trading.equals("Stock Trading")) {*

*System.out.println("Trader has selected Stock Trading option");*

*}****else******if****(trading.equals("Electronic Trading")) {*

*System.out.println("Trader has selected Electronic Trading option");*

In JDK 1.7

**switch** (trading) {

**case** "Stock Trading":

       System.*out*.println("Trader has selected Stock Trading option");

**break**;

**case** "Electronic Trading":

       System.*out*.println("Trader has selected Electronic Trading option");

**break**;

**default**: System.out.println("Invalid");

[equals() and hashcode()](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) method from java.lang.String is used in comparison, which is case-sensitive. Benefit of using String in switch is that, Java compiler can generate more efficient code than using nested if-then-else statement

**3)Try with resource feature –**

Before JDK 7, we need to use a [finally block](http://javarevisited.blogspot.sg/2012/11/difference-between-final-finally-and-finalize-java.html), to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly, for example while reading files and streams, we need to close them into finally block, which result in lots of boiler plate and messy code,

in Java 7, you can use try-with-resource feature to automatically close resources, which implements AutoClosable and Closeable interface e.g. Streams, Files, Socket handles, database connections etc. JDK 7 introduces a try-with-resources statement, which ensures that each of the resources in try(resources) is closed at the end of the statement by calling close() method of AutoClosable.

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

**try (FileInputStream fin = new FileInputStream("info.xml");** BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fin));)

{

**if** (br.ready()) {

String line1 = br.readLine(); System.out.println(line1);

}

}

**catch** (FileNotFoundException ex)

{

System.out.println("Info.xml is not found");

}

**catch** (IOException ex)

{

System.out.println("Can't read the file");

}

}

4)Underscore in numeric literals

In JDK 7, you could insert underscore(s) '\_' in between the digits in an numeric literals (integral and floating-point literals) to improve readability. This is especially valuable for people who uses large numbers in source files, may be useful in finance and computing domains. For example,

int billion **=** **1\_000\_000\_000**; // 10^9

long creditCardNumber **=** **1234\_4567\_8901\_2345L**; //16 digit number

long ssn **=** **777\_99\_8888L**;

double pi **=** **3.1415\_9265**;

float pif **=** **3.14\_15\_92\_65f**;

Invalid ones –

double pi **=** **3.\_1415\_9265**; // underscore just after decimal point

long creditcardNum **=** **1234\_4567\_8901\_2345\_L**; //underscore at the end of number

long ssn **=** **\_777\_99\_8888L**; //undersocre at the beginning

5)Catchng multiple exceptions in single catch block

Try {

......

} **catch**(**ClassNotFoundException**|**SQLException** ex) {

ex.printStackTrace();

}

By the way, just remember that Alternatives in a [multi-catch statement](http://javarevisited.blogspot.sg/2011/07/jdk7-multi-cache-block-example-tutorial.html)cannot be related by sub classing. For example a multi-catch statement like below will throw compile time error :

**try** {

......

} **catch** (**FileNotFoundException** | **IOException** ex) {

ex.printStackTrace();

}

Alternatives in a multi-catch statement cannot be related by sub classing, it will throw error at compile time :  
java.io.FileNotFoundException is a subclass of alternative java.io.IOException  
        at Test.main(Test.java:18)

6)Binary literals with prefix 0b

In JDK 7, you can express literal values in binary with prefix '0b' (or '0B') for integral types (byte, short, int and long),

int mask **=** **0**b01010000101;

7)JDK7 has new file API – Java NIO2.0-

### Java SE 7 introduced java.nio.file package and its related package, java.nio.file.attribute, provide comprehensive support for file I/O and for accessing the default file system. It also introduced the Path class which allow you to represent any path in operating system. 8) G1 Garbage Collector

### JDK 7 introduced a new Garbage Collector known as G1 Garbage Collection, which is short form of garbage first. It's said that G1 is quite predictable and provides greater through put for memory intensive applications. G1 garbage collector performs clean-up where there is most garbage. To achieve this it split [Java heap memory](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html) into multiple regions as opposed to 3 regions in the prior to Java 7 version (new, old and permgen space). 9) More Precise Rethrowing of Exception – (not clear ☹ )

From JDK 7 onwards you can be more precise while declaring type of Exception in [throws clause](http://java67.blogspot.sg/2012/10/difference-between-throw-vs-throws-in.html) of any method.  
  
  
public void precise() throws ParseException, IOException {

try {

new FileInputStream("abc.txt").read();

new SimpleDateFormat("ddMMyyyy").parse("12-03-2014");

} catch (Exception ex) {

System.out.println("Caught exception: " + ex.getMessage());

throw ex;

}

}

The Java SE 7 compiler allows you to specify the exception types ParseException and IOException in the throws clause in the preciese() method declaration because you can re-throw an exception that is a super-type of any of the types declared in the throws, we are throwing java.lang.Exception, which is super class of all [checked Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html)

Refer - http://javarevisited.blogspot.com/2014/04/10-jdk-7-features-to-revisit-before-you.html

1. OOPs concepts

**Is Java a pure object oriented language? if not why?** ([answer](http://java67.blogspot.com/2014/03/is-java-pure-object-oriented-programming-language.html))  
Java is not a pure object-oriented programming language e.g. there are many things you can do without objects e.g. static methods. Also, primitive variables are not objects in Java.

1. **Write your own hashmap implementation**

http://www.javamadesoeasy.com/2015/02/hashmap-custom-implementation.html

1. **What is cloning , how do you clone an object?**

**Cloning** is a process of creating an exact copy of an existing object in the memory. In java, **clone()** method of **java.lang.Object** class is used for cloning process. This method creates an exact copy of an object on which it is called through **field-by-field assignment** and returns the reference of that object. Not all the objects in java are eligible for cloning process. The objects which implement **Cloneable interface** are only eligible for cloning process. Cloneable interface is a [marker interface](http://javaconceptoftheday.com/marker-interface-java/) which is used to provide the marker to cloning process.

1) Let the class, which supports cloning implements Cloneable interface. (failure to do this will result in CloneNotSupportedException).  
  
2) Override protected clone() method from java.lang.Object class.  
  
3) In overridden clone(), first call super.clone() to get the shallow copy of object.  
  
4) If your class contains any Collection or Mutable object, than [deep copy](http://java67.blogspot.sg/2013/05/difference-between-deep-copy-vs-shallow-cloning-java.html) of those field. Like in our example, Programmer class contains List in it's certification field, when super.clone() will return, both original and cloned object will point to same object. To fix this, we reassign certification fields of clone object by explicitly copying data, as shown in following line :

clone.certifications = new ArrayList(certifications); //deep copying

5) Depending upon object, you may call it's clone method e.g. in case of java.util.Date or recursively copy it's data into new field. This whole process is known as deep copying

class Programmer implements Cloneable{

 private String name;

private int age;

 private List certifications ;

@Override protected Programmer clone() { Programmer clone = null; try{ clone = (Programmer) super.clone(); clone.certifications = new ArrayList(certifications); //deep copying

 }

catch(CloneNotSupportedException cns){ logger.error("Error while cloning programmer", cns); } return clone; }   
  
  
  
<http://javarevisited.blogspot.com/2015/01/java-clone-tutorial-part-2-overriding-with-mutable-field-example.html#ixzz4ejLEvXfx>

1. **Shallow copy and deep copy**

Refer-http://javaconceptoftheday.com/difference-between-shallow-copy-vs-deep-copy-in-java/

The default version of clone() method creates the shallow copy of an object. The shallow copy of an object will have exact copy of all the fields of original object. If original object has any references to other objects as fields, then only references of those objects are copied into clone object, copy of those objects are not created. That means any changes made to those objects through clone object will be reflected in original object or vice-versa. Shallow copy is not 100% disjoint from original object

Deep Copy

*class Student implements Cloneable*

*{*

*int id;*

*String name;*

*Course course;*

*public Student(int id, String name, Course course)*

*{*

*this.id = id;*

*this.name = name;*

*this.course = course;*

*}*

*//Overriding clone() method to create a deep copy of an object.*

*protected Object clone() throws CloneNotSupportedException*

*{*

*Student student = (Student) super.clone();*

*student.course = (Course) course.clone();*

*return student;*

*}*

*}*

**Course Class**

class Course implements Cloneable

{

    String subject1;

    String subject2;

    String subject3;

    public Course(String sub1, String sub2, String sub3)

    {

        this.subject1 = sub1;

        this.subject2 = sub2;

        this.subject3 = sub3;

    }

    protected Object clone() throws CloneNotSupportedException

    {

        return super.clone();

    }

}

## Shallow Copy Vs Deep Copy In Java :

Below is the list of differences between shallow copy and deep copy in java.

|  |  |
| --- | --- |
| **Shallow Copy** | **Deep Copy** |
| Cloned Object and original object are not 100% disjoint. | Cloned Object and original object are 100% disjoint. |
| Any changes made to cloned object will be reflected in original object or vice versa. | Any changes made to cloned object will not be reflected in original object or vice versa. |
| Default version of clone method creates the shallow copy of an object. | To create the deep copy of an object, you have to override clone method. |
| Shallow copy is preferred if an object has only primitive fields. | Deep copy is preferred if an object has references to other objects as fields. |
| Shallow copy is fast and also less expensive. | Deep copy is slow and very expensive. |

1. **Is map part of collection framework?**

Collection has subinterfaces such as Set, List and Queue. But Map is altogether a separate Interface

It's part of the collection framework but it doesn't implement the java.util.Collection interface. Maps work with key/value pairs, while the other collections work with just values. "the Collections Framework" is a general term that includes both Maps and Collections. A [HashMap](http://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html) is a Map, and is part of the Collections Framework, but is not a Collection. An [ArrayList](http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \t "_new" \o "Java API) is a List, and is part of the Collections Framework, and is a Collection.

|  |  |
| --- | --- |
|  | Because Map is three collections: Keys, values and key-value pairs.  It's not easy to decide which one should be the default to iterate over, so it implements the interface three times (keySet(), values() and entrySet()) |

1. Linked list vs Arraylist – We use linked list when elements are to be added/removed from the middle of the collection, then it is difficult to use array list, since we need to move each element to next position and insert the element at right position, whereas if we use LL, then it can be done easily by just updating the neighbor nodes.Some of the methods of LL are addFirst(),addlast(),removeFirst(),removeLast(),getFirst(),getLast()

Drawback of LL as compared to AL is that it is slow when we want to access elements at random position.

13) **What is the difference between Spring AOP and AspectJ AOP?**

AspectJ is the industry-standard implementation for Aspect Oriented Programming whereas Spring implements AOP for some cases. Main differences between Spring AOP and AspectJ are:

* Spring AOP is simpler to use than AspectJ because we don’t need to worry about the weaving process.
* Spring AOP supports AspectJ annotations, so if you are familiar with AspectJ then working with Spring AOP is easier.
* Spring AOP supports only proxy-based AOP, so it can be applied only to method execution join points. AspectJ support all kinds of pointcuts.
* One of the shortcoming of Spring AOP is that it can be applied only to the beans created through Spring Context.

**14)how does hashmap and linkedhashmap handles collisions?**

1) HashMap handles collision by using linked list to store map entries ended up in same array location or bucket location.  
  
2) From Java 8 onwards, HashMap, ConcurrentHashMap, and LinkedHashMap will use the balanced tree in place of [linked list](http://java67.blogspot.com/2016/01/how-to-implement-singly-linked-list-in-java-using-generics-example.html) to handle frequently hash collisions. The idea is to switch to the balanced tree once the number of items in a hash bucket grows beyond a certain threshold. This will improve the worst case get() method performance from O(n) to O(log n).  
  
3) By switching from linked list to balanced tree for handling collision, the iteration order of HashMap will change. This is Ok because HashMap doesn't provide any guarantee on iteration order and any code which depends upon that are likely to break.  
  
4) Legacy class Hashtable which exists in JDK from Java 1 will not use the balanced binary tree to handle frequent hash collision to keep its iteration order intact. This was decided to avoid breaking many legacy Java application which depends upon iteration order of Hashtable.  
  
5) Apart from Hashtable, WeakHashMap and IdentityHashMap will also continue to use the linked list for handling collision even in the case of frequent collisions.  
  
6) Collision in [HashMap](http://java67.blogspot.com/2016/01/how-to-initialize-hashmap-with-values-in-java.html) is possible because hash function uses hashCode() of key object and equals() and hashCode() contract doesn't guarantee different hashCode for different objects.   
7) A collision will occur on Hashtable or HashMap when hashCode() method of two different key objects will return same values.

15)Programmatic vs declarative transaction management in spring

**Programmatic Transaction Management**

1. Allows you to manage transactions through programming in your source code.
2. This means *hardcoding* transaction logic between your business logic.
3. You use programming to manage transactions
4. Flexible, but difficult to maintain with large amount of business logic. Introduces boilerplate between business logic.
5. Preferred when relative less transaction logic is to be introduced.

**Declarative Transaction Management**

1. Allows you to manage transactions through configuration.
2. This means *separating* transaction logic with business logic.
3. You use annotations (Or XML files) to manage transactions.
4. Easy to maintain. Boilerplate is kept away from business logic.
5. Preferred when working with large amount of Transaction logic.

16)difference between @service and @component annotation

Annotation | Meaning |

+------------+-----------------------------------------------------+

| @Component | generic stereotype for any Spring-managed component |

| @Repository| stereotype for persistence layer |

| @Service | stereotype for service layer |

| @Controller| stereotype for presentation layer (spring-mvc) |

In Spring 2.0 and later, the @Repository annotation is a marker for any class that fulfills the role or stereotype (also known as Data Access Object or DAO) of a repository. Among the uses of this marker is the automatic translation of exceptions.

@Repository, @Service, and @Controller are specializations of @Component for more specific use cases, for example, in the persistence, service, and presentation layers, respectively.

<context:component-scan> only scans @Component and do not look

for @Controller, @Serviceand @Repository in general. They are scanned because they themselves are annotated with @Component.

Just take a look at @Controller, @Service and @Repository annotation definition

@Component

public @interface Service {

….

}

@Component

public @interface Repository {

….

}

@Component

public @interface Controller {

…

}

In addition to point out that this is an Annotation based Configuration, @Repository’s job is to catch Platform specific exceptions and re-throw them as one of Spring’s unified unchecked exception. And for this, we’re provided with PersistenceExceptionTranslationPostProcessor,

@Component – Indicates a auto scan component.

@Repository – Indicates DAO component in the persistence layer.

@Service – Indicates a Service component in the business layer.

@Controller – Indicates a controller component in the presentation layer.

17)what is jdbctemplate in spring and how to use it?

Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries. It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

## Problems of JDBC API

The problems of JDBC API are as follows:

* We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
* We need to perform exception handling code on the database logic.
* We need to handle transaction.
* Repetition of all these codes from one to another database logic is a time consuming task.

## Advantage of Spring JdbcTemplate

Spring JdbcTemplate eliminates all the above mentioned problems of JDBC API. It provides you methods to write the queries directly, so it saves a lot of work and time.

## JdbcTemplate class

It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of connection object etc. So it will not lead to any problem if you forget to close the connection.

It handles the exception and provides the informative exception messages by the help of exception classes defined in the **org.springframework.dao** package.

19)JPA - Hibernate is a JPA provider. JPA is a specification for accessing, persisting and managing the data between Java objects and the relational database. **JPA is just guidelines to implement the Object Relational Mapping (ORM)  and there is no underlying code for the implementation.** Where as, Hibernate is the actual implementation of JPA guidelines

**20)difference between fielsystemresource and classpathresource**

* FileSystemResource can locate given spring bean configuration file from the specified path of the file system. Here we can pass either absolute path or relative path.

FileSystemResourceres = new FileSystemResource("src/com/example/cfgs/applicationContext.xml");

* ClassPathResource can locate given spring bean configuration file from directories or jar files that are added to classpath/Build path.

ClassPathResource res = new ClassPathResource("applicationContext.xml");

21)spring MVC v struts mvc?

1)Struts 2 has built-in Ajax support so there is no need to import a third-party library. Although SpringMVC does not have native Ajax support, this just gives a more seasoned developer the ability to choose its Ajax library.

2) SpringMVC applications must be constantly monitored for improvements and upgrades are a must. This is less important for Struts 2 because the underlying framework is not changing much.

3) [Spring](http://www.technorati.com/tag/Spring) provides a very clean division between controllers, [JavaBean](http://www.technorati.com/tag/Javabean) models, and views.

4). [Spring’s MVC](http://www.technorati.com/tag/Spring+MVC) is very flexible. Unlike Struts, which forces your Action and Form objects into concrete inheritance (thus taking away your single shot at concrete inheritance in Java), [Spring MVC](http://www.technorati.com/tag/Spring+MVC) is entirely based on interfaces. Furthermore, just about every part of the [Spring MVC](http://www.technorati.com/tag/Spring+MVC) framework is configurable via plugging in your own interface. Of course we also provide convenience classes as an implementation option.

5)  [Spring MVC](http://www.technorati.com/tag/Spring+MVC) is truly view-agnostic. You don’t get pushed to use JSP if you don’t want to; you can use Velocity, [XLST](http://www.technorati.com/tag/XSLT) or other view technologies. If you want to use a custom view mechanism – for example, your own templating language – you can easily implement the Spring View interface to integrate it.

22)new features in spring 3.0?

Spring 3.0 makes the entire spring code base to take advantage of the [Java 5.0](http://www.javabeat.net/2010/07/java-5-0/) technology.  The notable Java 5 features like [Generics](http://www.javabeat.net/2007/08/generics-in-java-5-0/), **Varargs**, Annotations and other improvements has been extensively implemented with the Spring 3.0 release.

* Full-scale [**REST**](http://javabeat.net/recommends/rest-with-spring/) support in Spring MVC, including **Spring MVC** controllers that respond to REST-style URLs with XML, **JSON**, RSS or any other appropriate response
* A new [expression language](http://www.javabeat.net/2011/02/introduction-to-spring-expression-language-spel/) that brings Spring dependency injection to a new level by enabling injection of values from a verity of sources, including beans and system properties.

**Spring Expression Language-**Basically helps you in querying an object at run time, instead of having to depend only on configuration files. You can use it for both XML and annotation based configurations. It supports a wide range from relational expressions to accessing properties of a class or method.

For eg if I want to specify current Date for a bean’s date property, it can be done this way

1. **<bean** id="employee" class="com.test.Employee"**>**
2. **<property** name="joinDate" value="#{new java.util.Date()}" **/>**
3. **</bean>**

Or

1. **public** **class** Employee{
2. @Value("#{Rajiv}")
3. **private** String empName;
4. **private** Date joinDate;
5. @Value("#{new java.util.Date()}")
6. **public** **void** setJoinDate(Date joinDate) {
7. **this**.joinDate=joinDate;
8. }
9. }

* New annotations for Spring MVC, including @CookieValue and @Request–Header, to pull values from cookies and request headers, respectively.
* A new XML namespace for easing configuration of **Spring MVC**.
* The **Object-to-XML (OXM)** mapping functionality from the Spring Web services project has been moved into the core Spring framework.

23)dialect? - Dialect means "the variant of a language".  Hibernate uses "dialect" configuration to know which database you are using so that it can switch to the database specific SQL generator code. Since databases have proprietary extensions/native SQL variations, and set/sub-set of SQL standard implementations

A **hibernate dialect** gives information to the framework of how to **convert** hibernate queries(HQL) into native SQL queries.

Things such as data types for example vary across databases (e.g. in Oracle You might put an integer value in a number field and in SQL Server use an int field). Or database specific functionality - selecting the top n rows is different depending on the database. The dialect abstracts this so you don't have to worry about it.

24)benefits of immutable class

* Immutable objects are automatically thread-safe, the overhead caused due to use of synchronisation is avoided.
* Once created the state of the immutable object can not be changed so there is no possibility of them getting into an inconsistent state.
* The references to the immutable objects can be easily shared or cached without having to copy or clone them as there state can not be changed ever after construction.
* Immutable objects are good **Map** keys and **Set** elements, since these typically do not change once created.

Immutability can have a performance cost, since when an object cannot be mutated we need to copy it if we want to write to it. When you care a lot about performance (e.g. programming a game) it may be necessary to use a mutable object.

25) **Q2) How to create an immutable class?**

**Ans)** To create an immutable class following steps should be followed:

1. Create a final class.
2. Set the values of properties using constructor only.
3. Make the properties of the class final and private
4. Do not provide any setters for these properties.
5. Do not change the state of the objects in any methods of the class
6. If the instance fields include references to mutable objects, don't allow those objects to be changed:
   1. Don't provide methods that modify the mutable objects.
   2. Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

public final class FinalPersonClass {

private final String name;

private final int age;

public FinalPersonClass(final String name, final int age) {

this.name = name;

this.age = age;

}

public int getAge() {

return age;

}

public String getName() {

return name;

}

}

26)concurrenthashmap

26) **What are the exceptions thrown by the Spring DAO classes ?**

Spring DAO classes throw exceptions which are subclasses of DataAccessException(org.springframework.dao.DataAccessException).Spring provides a convenient translation from technology-specific exceptions like SQLException to its own exception class hierarchy with the DataAccessException as the root exception. These exceptions wrap the original exception.

27)Design patterns

28)Lazy loading - FetchType.LAZY = Doesn’t load the relationships unless explicitly “asked for” via getter  
FetchType.EAGER = Loads ALL relationships

[How to work with wait(), notify() and notifyAll() in Java? - HowToDoInJava](https://howtodoinjava.com/java/multi-threading/wait-notify-and-notifyall-methods/#:~:text=The%20highest%20priority%20thread%20will%20run%20first%20in,like%20this%3A%20synchronized%28lockObject%29%20%7B%20establish_the_condition%3B%20lockObject.notifyAll%20%28%29%3B%20%7D)

#### If a thread receives a notification, is it guaranteed that the condition is set correctly?

Simply, no. Prior to calling the wait() method, a thread should always test the condition while holding the synchronization lock. Upon returning from the wait() method, the thread should always retest the condition to determine if it should wait again. This is because another thread can also test the condition and determine that a wait is not necessary — processing the valid data that was set by the notification thread.

This is a common case when multiple threads are involved in the notifications. More particularly, the threads that are processing the data can be thought of as consumers; they consume the data produced by other threads. There is no guarantee that when a consumer receives a notification that it has not been processed by another consumer. As such, when a consumer wakes up, it cannot assume that the state it was waiting for is still valid. It may have been valid in the past, but **the state may have been changed after the notify() method was called and before the consumer thread woke up.** Waiting threads must provide the option to check the state and to return back to a waiting state in case the notification has already been handled**. This is why we always put calls to the wait() method in a loop.**

#### What happens when more than one thread is waiting for notification? Which threads actually get the notification when the notify() method is called?

It depends on many factors.Java specification doesn’t define which thread gets notified. In runtime, which thread actually receives the notification varies based on several factors, including the implementation of the Java virtual machine and scheduling and timing issues during the execution of the program. There is no way to determine, even on a single processor platform, which of multiple threads receives the notification.

Just like the notify() method, the notifyAll() method does not allow us to decide which thread gets the notification: they all get notified. When all the threads receive the notification, it is possible to work out a mechanism for the threads to choose among themselves which thread should continue and which thread(s) should call the wait() method again.

#### Does the notifyAll() method really wake up all the threads?

Yes and no. All of the waiting threads wake up, but they still have to reacquire the object lock. So the threads do not run in parallel: they must each wait for the object lock to be freed. Thus, only one thread can run at a time, and only after the thread that called the notifyAll() method releases its lock.

#### Why would you want to wake up all of the threads if only one is going to execute at all?

There are a few reasons. For example, there might be more than one condition to wait for. Since we cannot control which thread gets the notification**, it is entirely possible that a notification wakes up a thread that is waiting for an entirely different condition**. By waking up all the threads, we can design the program so that the threads decide among themselves which thread should execute next. **Another option could be when producers generate data that can satisfy more than one consumer**. Since it may be difficult to determine how many consumers can be satisfied with the notification, an option is to notify them all, allowing the consumers to sort it out among themselves.

String s = new String("Test");  
   
does not  put the object in String pool , we need to call String.intern() method which is used to put  them into String pool explicitly. its only when you create String object as String literal e.g. String s = "Test" Java automatically put that into String pool.

**we can not override static method**, we can only hide static method in Java. Creating static method with same name and mehtod signature is called Method hiding in Java

you cannot override static method in Java because [method overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) is based upon dynamic binding at runtime and static methods are bonded using [static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time.

**public** **class** CanWeOverrideStaticMethod {  
    
    **public** **static** **void** main(**String** args[]) {  
        
        Screen scrn = **new** ColorScreen();  
        
        *//if we can  override static , this should call method from Child class*  
        scrn.show(); *//IDE will show warning, static method should be called from classname*  
        
    }    
    
}  
  
**class** Screen{  
    
    */\*  
     \* public static method which can not be overridden in Java  
     \*/*  
    **public** **static** **void** show(){  
        **System**.out.printf("Static method from parent class");  
    }  
}  
**class** ColorScreen **extends** Screen{  
    */\*  
     \* static method of same name and method signature as existed in super  
     \* class, this is not method overriding instead this is called  
     \* method hiding in Java  
     \*/*  
    **public** **static** **void** show(){  
        **System**.err.println("Overridden static method in Child Class in Java");  
    }  
}  
**Output:**  
Static method from parent **class**  
It means if you use Parent class's type to call static method, original static will be called from patent class, on ther other hand if you use Child class's type to call static method, method from child class will be called. 

http://www.geeksforgeeks.org/prevent-singleton-pattern-reflection-serialization-cloning/

***/\*\*  
\* Singleton pattern example using Java Enumj  
\*/***

**public** **enum** EasySingleton{  
    INSTANCE;  
}

**Singleton example with double checked locking**

***/\*\*  
\* Singleton pattern example with Double checked Locking  
\*/***

**public** **class** DoubleCheckedLockingSingleton{  
     **private** **volatile** DoubleCheckedLockingSingleton INSTANCE;  
    
     **private** DoubleCheckedLockingSingleton(){}  
    
     **public** DoubleCheckedLockingSingleton getInstance(){  
         **if**(INSTANCE == **null**){  
            **synchronized**(DoubleCheckedLockingSingleton.**class**){  
                *//double checking Singleton instance*  
                **if**(INSTANCE == **null**){  
                    INSTANCE = **new** DoubleCheckedLockingSingleton();  
                }  
            }  
         }  
         **return** INSTANCE;  
     }  
}

**Singleton pattern with static factory method**

***/\*\*  
\* Singleton pattern example with static factory method  
\*/***  
  
**public** **class** Singleton{  
    *//initailzed during class loading*  
    **private** **static** **final** Singleton INSTANCE = **new** Singleton();  
    
    *//to prevent creating another instance of Singleton*  
    **private** Singleton(){}  
  
    **public** **static** Singleton getSingleton(){  
        **return** INSTANCE;  
    }  
}

**Singleton Design Pattern**  
  
  
Singletons are useful only when you need one instance of a class and it is undesirable to have more than one instance of a class.

Example - Property file reading.

Usually we deal with the property files , property files can be loaded once, and though out the application we can use the object without re instance creation. In this scenario we can make use of Singleton Design Pattern

Here is the code to make use of singleton pattern by reading property file

public class SingletonDesign{

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

SingleTon ton = SingleTon.getSingleTonObject();

System.out.println(ton.getProperties().get("name"));

System.out.println(ton.getProperties().get("blogname"));

ton.getProperties().setProperty("email", "something@gmail.com");

System.out.println(ton.getProperties().get("email"));

}

}

class SingleTon{

public static SingleTon singleTon=null;

public Properties properties=null;

public Properties getProperties() {

return properties;

}

public void setProperties(Properties properties) {

this.properties = properties;

}

private SingleTon(){

}

public static synchronized SingleTon getSingleTonObject(){

if(singleTon==null){

singleTon = new SingleTon();

Properties properties1 = new Properties();

try{

properties1.load(new FileInputStream("D:/propertiesFile.properties"));

singleTon.setProperties(properties1);

}catch(FileNotFoundException ex){

ex.printStackTrace();

}catch(IOException ioe){

ioe.printStackTrace();

}

return singleTon;

}

return singleTon;

}

public Object clone() throws CloneNotSupportedException{

throw new CloneNotSupportedException();

}

}

outcome

----------

suneel

http://suneel-javautils.blogspot.com

something@gmail.com  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. public void start(){
2. }
3. **What is the problem with below program?**
4. package com.journaldev.exceptions;
5. import java.io.IOException;
6. import javax.xml.bind.JAXBException;
7. public class TestException6 {
8. public static void main(String[] args) {
9. try {
10. foo();
11. } catch (IOException | JAXBException e) {
12. e = new Exception("");
13. e.printStackTrace();
14. }catch(Exception e){
15. e = new Exception("");
16. e.printStackTrace();
17. }
18. }
19. public static void foo() throws IOException, JAXBException{
21. }

}

The above program won’t compile because exception object in multi-catch block is final and we can’t change it’s value. You will get compile time error as “The parameter e of a multi-catch block cannot be assigned”.

**What is contextloaderlistener? -**to automate the creation of the ApplicationContext, so you don't have to write explicit code to do create it . ContextLoaderListener reads the Spring configuration file (with value given against contextConfigLocation in web.xml), parses it and loads the singleton bean defined in that config file. Similarly when we want to load prototype bean, we will use same webapplication context to load it.

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/applicationContext-\*.xml

classpath:applicationContext-business.xml

classpath:applicationContext-dummyChip.xml

classpath:applicationContext-dummyWebService.xml

</param-value>

</context-param>

FOR SPRING MVC APPLICATION

* There will be only one application context per web application. Apart from **ApplicationContext**, there can be multiple **WebApplicationContext** in a single web application. each DispatcherServlet associated with single WebApplicationContext.

Basically you can isolate your root application context and web application context using ContextLoaderListner.

The config file mapped with context param will behave as root application context configuration. And config file mapped with dispatcher servlet will behave like web application context.

In any web application we may have multiple dispatcher servlets, so multiple web application contexts.

But in any web application we may have only one root application context that is shared with all web application contexts.

For a more complex Spring application, where you have multiple DispatcherServlet defined, you can have the common Spring configuration files that are shared by all the DispatcherServlet defined in the ContextLoaderListener:

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/common-config.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>mvc1</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/mvc1-config.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet>

<servlet-name>mvc2</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/mvc2-config.xmll</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

|  |  |
| --- | --- |
| 43 | <servlet-mapping> |

|  |  |
| --- | --- |
| 44 | <servlet-name>webmvc1</servlet-name> |

|  |  |
| --- | --- |
| 45 | <url-pattern>/webmvc1</url-pattern> |

|  |  |
| --- | --- |
| 46 | </servlet-mapping> |

|  |  |
| --- | --- |
| 47 | <servlet-mapping> |

|  |  |
| --- | --- |
| 48 | <servlet-name>webmvc2</servlet-name> |

|  |  |
| --- | --- |
| 49 | <url-pattern>/webmvc2</url-pattern> |

|  |  |
| --- | --- |
| 50 | </servlet-mapping> |

</servlet>

A web application can define any number of DispatcherServlets. Each servlet will operate in its own namespace, loading its own application context with mappings, handlers, etc. Only the root application context as loaded by ContextLoaderListener, if any, will be shared.

So, you're sessionFactory will only be shared if it's defined in an app context loaded by the [ContextLoaderListener](http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/web/context/ContextLoaderListener.html), otherwise not.

What is load on startup?

load-on-startup can specify an (optional) integer value. If the value is 0 or greater, it indicates an order for servlets to be loaded, servlets with higher numbers get loaded after servlets with lower numbers.

**Short Answer**: value >= 0 means that the servlet is loaded when the web-app is deployed or when the server starts. value < 0 : servlet is loaded whenever the container feels like.

The load-on-startup element indicates that this servlet should be loaded (instantiated and have its init() called) on the startup of the web application.

JUnit 4 added annotations to the framework and eliminated the need to extend TestCase. You can direct both the lifecycle events and other aspects of the test execution with the provided annotations.

|  |  |  |
| --- | --- | --- |
| **Annotation** | **Parameters** | **Use** |
| @After | None | Method will be executed after each test method (similar to the tearDown() method in JUnit 3.x). Multiple methods may be tagged with the @After annotation, however no order is guaranteed. |
| @AfterClass | None | Method will be executed after all of the test methods and teardown methods have been executed within the class. Multiple methods may be tagged with the @AfterClass annotation, however no order is guaranteed. |
| @Before | None | Method will be executed before each test method (similar to the setUp() method in JUnit 3.x). Multiple methods may be tagged with the @Before annotation, however no order is guaranteed. |
| @BeforeClass | None | Executed before any other methods are executed within the class. Multiple methods  s may be tagged with the @BeforeClass annotation, however no order is guaranteed. |

|  |  |  |
| --- | --- | --- |
| @Test | * Class(optional) * Timeout(optional) | Used to indicate a test method. Same functionality as naming a method public void testXYZ() in JUnit 3.x. |

EasyMock Mock Object Lifecycle, Continued

Stage Description

Create Mock This phase creates the mock object.

Expect This phase records the expected behaviors of the mock object. These will be verified at the end.

Replay Replays the previously recorded expectations.(activating mocks)

Verify In order for a test to pass, the expected behaviors must have been executed. The verify phase confirms the execution of the expected calls.

OOPS concepts refer below:

<http://www.w3resource.com/java-tutorial/java-object-oriented-programming.php>

public class Demo {

public Demo() {

System.out.println("Hi!");

}

public static void main(String[] args) throws Exception {

Class clazz = Class.forName("test.Demo");

Demo demo = (Demo) clazz.newInstance();

}

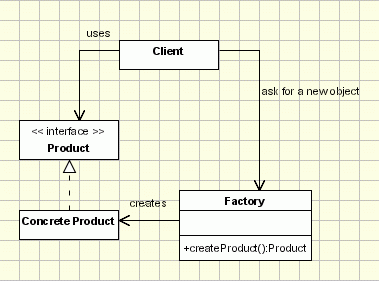
calling [Class.forName(String)](http://java.sun.com/javase/6/docs/api/java/lang/Class.html" \l "forName%28java.lang.String%29) returns the*Class*object associated with the class with the given string name

calling [clazz.newInstance()](http://java.sun.com/javase/6/docs/api/java/lang/Class.html#newInstance%28%29) creates a new instance of the class represented by this*Class* object The class is instantiated as if by a*new*expression with an empty argument list.

Dependency Inversion principle:

Rather than having the higher-level module directly depending on lower-level modules to carry out their responsibilities, this principle instead makes the higher-level module rely on an ‘abstraction’ or an ‘abstract interface’ representing lower-level module. The actual implementation of lower level module can then vary. As long as the lower-level module’s implementation is accessible to the higher-level module via the abstract interface, the higher-level module is able to invoke it.

Factory Pattern – Refer http://www.oodesign.com/factory-pattern.html



The implementation is really simple

* The client needs a product, but instead of creating it directly using the new operator, it asks the factory object for a new product, providing the information about the type of object it needs.
* The factory instantiates a new concrete product and then returns to the client the newly created product(casted to abstract product class).
* The client uses the products as abstract products without being aware about their concrete implementation.

Whenever we create object using new() we violate **principle of programming for interface rather than implementation** which eventually result in inflexible code and difficult to change in maintenance. By using Factory design pattern in Java we get rid of this problem.

Here is complete **code example of Factory pattern in Java**:

**interface** Currency {

       String getSymbol();

}

// Concrete Rupee Class code

**class** Rupee **implements** Currency {

       @Override

**public** String getSymbol() {

**return** "Rs";

       }

}

// Concrete SGD class Code

**class** SGDDollar **implements** Currency {

       @Override

**public** String getSymbol() {

**return** "SGD";

       }

}

// Concrete US Dollar code

**class** USDollar **implements** Currency {

       @Override

**public** String getSymbol() {

**return** "USD";

       }

}

// Factroy Class code

**class** CurrencyFactory {

**public** **static** Currency createCurrency (String country) {

**if** (country. equalsIgnoreCase ("India")){

**return** **new** Rupee();

       }**else** **if**(country. equalsIgnoreCase ("Singapore")){

**return** **new** SGDDollar();

       }**else** **if**(country. equalsIgnoreCase ("US")){

**return** **new** USDollar();

        }

**throw** **new** IllegalArgumentException("No such currency");

       }

}

// Factory client code

**public** **class** Factory {

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

              String country = args[0];

              Currency rupee = CurrencyFactory.*createCurrency*(country);

              System.*out*.println(rupee.getSymbol());

       }

}

1*) Factory method design pattern* decouples the calling class from the target class, which result in less coupled and highly cohesive code?

E.g.: JDBC is a good example for this pattern; application code doesn't need to know what database it will be used with, so it doesn't know what database-specific driver classes it should use. Instead, it uses factory methods to get Connections, Statements, and other objects to work with. Which gives you flexibility to change your back-end database without changing your DAO layer in case you are using ANSI SQL features and not coded on DBMS specific feature?

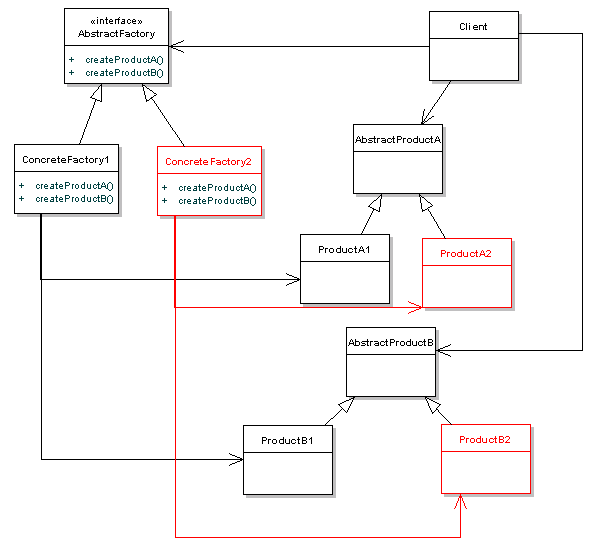
All the wrapper classes like Integer, Boolean etc, in Java uses this pattern to evaluate the values using valueOf() method.   
3. java.nio.charset.Charset.forName(), java.sql.DriverManager#getConnection(), java.net.URL.openConnection(), java.lang.Class.newInstance(), java.lang.Class.forName() are some of their example where factory method design pattern has been used.

2) Factory pattern in Java enables the subclasses to provide extended version of an object, because creating an object inside factory is more flexible than creating an object directly in the client. Since client is working on interface level any time you can enhance the implementation and return from Factory.

3) Another benefit of using *Factory design pattern in Java* is that it encourages [consistency in Code](http://javarevisited.blogspot.com/2011/09/code-review-checklist-best-practice.html) since every time object is created using Factory rather than using different constructor at different client side.

1. Code written using Factory design pattern in Java is also [easy to debug](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html) and troubleshoot because you have a centralized method for object creation and every client is getting object from same place.
2. By using Factory method design pattern client is completely decoupled with object creation code, which enforces [Encapsulation](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) and result is loosely coupled and highly cohesive system.

*Abstract Factory Pattern-* **Provides an interface for creating families of related or dependent objects without specifying their concrete classes.**



The **AbstractFactory**defines the interface that all of the concrete factories will need to implement in order to product **Product**s. **ConcreteFactoryA**and **ConcreteFactoryB**have both implemented this interface here, creating two seperate families of product. Meanwhile, **AbstractProductA**and **AbstractProductB**are interfaces for the different types of product. Each factory will create one of each of these AbstractProducts.

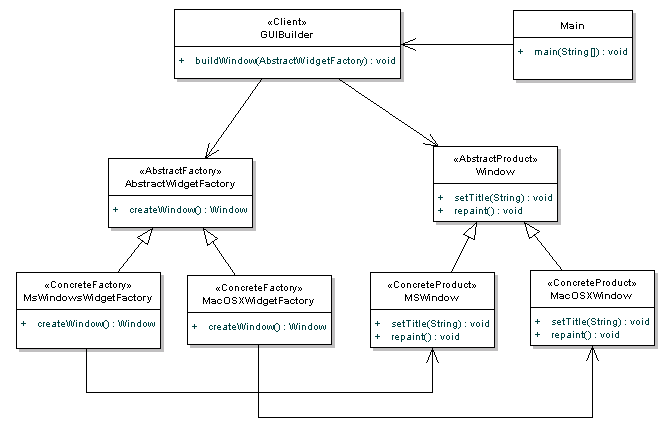
The **Client**deals with **AbstractFactory**, **AbstractProductA**and **AbstractProductB**. It doesn't know anything about the implementations. The actual implementation of **AbstractFactory**that the **Client**uses is determined at runtime.

Advantage

The client is totally decoupled from the concrete products. Also, new product families can be easily added into the system, by just adding in a new type of **ConcreteFactory**that implements **AbstractFactory**, and creating the specific Product implementations.

**Where Would I Use This Pattern?**

The pattern is best utilised when your system has to create multiple families of products or you want to provide a library of products without exposing the implementation details



//Our AbstractProduct

public interface Window{

public void setTitle(String text);

public void repaint();

}

Let's create two implementations of the Window, as our **ConcreteProduct**s. One for Microsoft Windows and one for Mac OSX

//ConcreteProductA1

public class MSWindow implements Window{

public void setTitle(){

//MS Windows specific behaviour

}

public void repaint(){

//MS Windows specific behaviour

}

}

//ConcreteProductA2

public class MacOSXWindow implements Window{

public void setTitle(){

//Mac OSX specific behaviour

}

public void repaint(){

//Mac OSX specific behaviour

}

}

Now we need to provide our factories. First we'll define our **AbstractFactory**. For this example, let's say they just create Windows:

//AbstractFactory

public interface AbstractWidgetFactory{

public Window createWindow();

}

Next we need to provide **ConcreteFactory**implementations of these factories for our two operating systems. First for MS Windows:

//ConcreteFactory1

public class MsWindowsWidgetFactory{

//create an MSWindow

public Window createWindow(){

MSWindow window = new MSWindow();

return window;

}

}

And for MacOSX:

//ConcreteFactory2

public class MacOSXWidgetFactory{

//create a MacOSXWindow

public Window createWindow(){

MacOSXWindow window = new MacOSXWindow();

return window;

}

}

Finally we need a client to take advantage of all this functionality.

//Client

public class GUIBuilder{

public void buildWindow(AbstractWidgetFactory widgetFactory){

Window window = widgetFactory.createWindow();//composition

window.setTitle("New Window");

}

}

Of course, we need some way to specify which type of AbstractWidgetFactory to our GUIBuilder. This is usually done with a switch statement similar to the code below:

public class Main{

public static void main(String[] args){

GUIBuilder builder = new GUIBuilder();

AbstractWidgetFactory widgetFactory = null;

//check what platform we're on

if(Platform.currentPlatform()=="MACOSX"){

widgetFactory = new MacOSXWidgetFactory();

} else {

widgetFactory = new MsWindowsWidgetFactory();

}

builder.buildWindow(widgetFactory);

}

}

Disadvantage : there's one thing that annoys me - someone has to determine what type of factory the client is dealing with at runtime

Another example of Abstract factory pattern-

http://www.avajava.com/tutorials/lessons/abstract-factory-pattern.html

Difference between factory and Abstract factory pattern

* In short Abstract Factory design pattern provides abstraction over Factory pattern itself while Factory design pattern provides abstraction over products.
* AbstractFactory pattern uses composition to delegate responsibility of creating object to another class while Factory design pattern uses [inheritance](http://javarevisited.blogspot.sg/2012/10/what-is-inheritance-in-java-and-oops-programming.html) and relies on derived class or sub class to create [object](http://javarevisited.blogspot.sg/2012/12/what-is-object-in-java-or-oops-example.html)

in Abstract Factory, there’s a separate class dedicated to create a family of related/dependent Products and its (any concrete subclass factory) object can be passed to the client which uses it (composition). Here the Client gets a different object (concrete factory) to create the Products, instead of creating itself (e.g. using factoryMethod() and forcing inheritance), and thus uses composition.

* Factory Method is used to create one product only but Abstract Factory is about creating families of related or dependent products.
* Abstract Factory is one level higher in abstraction than Factory Method. Factory Method abstracts the way objects are created, while Abstract Factory also abstracts the way factories are created which in turn abstracts the way objects are created..

Why spring uses only unchecked exceptions?

Answer - That's why Spring, for example, uses RuntimeExceptions. The API still declares which exceptions may be thrown so you can catch those you're interested in -- those you know how to handle -- and the rest are bubbled upwards all the way until the layer of your code that knows what to do with the exception. And you don't have to do "try { ... } catch (Exception e) { throw e; }" all the time.

unchecked exceptions rescue developers from adding unnecessary catch blocks. If you want still you can catch desired unchecked exception and leave the rest of them which is not the case with checked exceptions.

Que – Sort an array of integers by characters. For example )Input is [1,34,3,98,9,76,45,4] and the output should be 998764543431 –

Given two numbers X and Y, how should myCompare() decide which number to put first – we compare two numbers XY (Y appended at the end of X) and YX (X appended at the end of Y). If XY is larger, then X should come before Y in output, else Y should come before

 let X and Y be 542 and 60. To compare X and Y, we compare 54260 and 60542. Since 60542 is greater than 54260, we put Y first.

**Answer** - to solve this is to pad the integers to the same size by repeating the digits then sort using these repeated integers as a sort key.(pad to length=max length\*2)

So while comparing the elements 98 and 9, in the comparator logic we will compare 9999 and 9898 so will return “9” is greater than “98”

Refer -https://rosettacode.org/wiki/Largest\_int\_from\_concatenated\_ints#Java

**import** ***java.util.\****;  
   
**public** **class** IntConcat **{**  
   
 **private** **static** [Comparator](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Comparator.html)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)> sorter = **new** [Comparator](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Comparator.html)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)>**(){**  
 @[Override](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Override.html)  
 **public** **int** compare**(**[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html) o1, [Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html) o2**){**  
 [String](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) o1s = o1.**toString()**;  
 [String](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) o2s = o2.**toString()**;  
   
 **if(**o1s.**length()** == o2s.**length()){**  
 **return** o2s.**compareTo(**o1s**)**;  
 **}**  
   
 **int** mlen = [Math](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html).**max(**o1s.**length()**, o2s.**length())**;  
 **while(**o1s.**length()** < mlen \* **2)** o1s += o1s;  
 **while(**o2s.**length()** < mlen \* **2)** o2s += o2s;  
   
 **return** o2s.**compareTo(**o1s**)**;  
 **}**  
 **}**;  
   
 **public** **static** [String](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) join**(**[List](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+List)<?> things**){**  
 [String](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) output = "";  
 **for(**[Object](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+Object) obj:things**){**  
 output += obj;  
 **}**  
 **return** output;  
 **}**  
   
 **public** **static** **void** main**(**[String](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html)**[]** args**){**  
 [List](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+List)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)> ints1 = **new** [ArrayList](http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)>**(**[Arrays](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Arrays.html).**asList(1**, **34**, **3**, **98**, **9**, **76**, **45**, **4))**;  
   
 [Collections](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collections.html).**sort(**ints1, sorter**)**;  
 [System](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).**out**.**println(**join**(**ints1**))**;// output should be -998764543431  
   
 [List](http://www.google.com/search?sitesearch=java.sun.com&q=allinurl%3Aj2se%2F1+5+0%2Fdocs%2Fapi+List)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)> ints2 = **new** [ArrayList](http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html)<[Integer](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Integer.html)>**(**[Arrays](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Arrays.html).**asList(54**, **546**, **548**, **60))**;  
   
 [Collections](http://java.sun.com/j2se/1.5.0/docs/api/java/util/Collections.html).**sort(**ints2, sorter**)**;  
 [System](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/System.html).**out**.**println(**join**(**ints2**))**;  
 **}**  
**}**

convert number to words

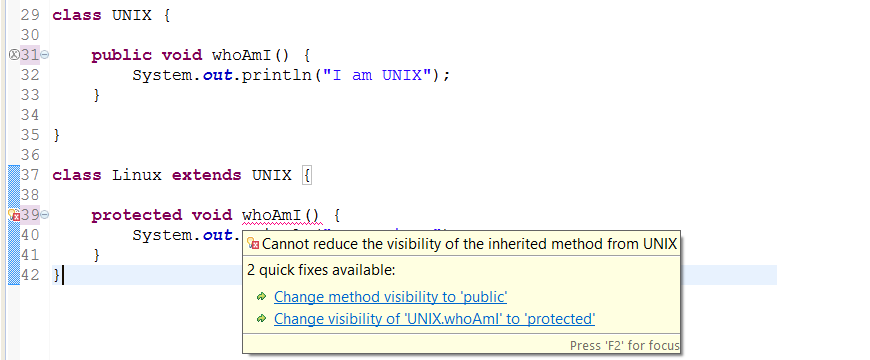
<https://gist.github.com/madan712/be79a395424bf773be87>

Rule of overriding

* 1. The **return type of overriding method can be child class of return type declared in overridden method**.

|  |
| --- |
| public class SuperClass {      //Overriden method      public Number sum(Integer a, Integer b) {          return a + b;      }  }    class SubClass extends SuperClass {      //Overriding method      @Override      public Integer sum(Integer a, Integer b) {      //Integer extends Number; so it's valid          return a + b;      }  } |

* 1. **Overriding method can not reduce access of overridden method**

 **3) If all 4 constructors defined in super and derived class. E.g super without arg,super with arg,child with arg and child without arg then:-**

* + 1. **If I call new child(arg) then super no arg is invoked followed by child arg constructor**
    2. **If I call new child() then super no arg is invoked followed by child no arg**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Arrays.asList()**

String[] asset = {"equity", "stocks", "gold", "foreign exchange","fixed income", "futures", "options"};

 List<String> assetList = Arrays.asList(asset);

* 1. This method returns a [List view](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html)of underlying array.
  2. When you call Arrays.asList it does not return a java.util.ArrayList. It returns a java.util.Arrays$ArrayList which is an immutable list. You cannot add to it and you cannot remove from it.

 So you should do something like:

List newList = new ArrayList(Arrays.asList(array));

in order to make the newList modifiable.

* 1. List returned by this method would be fixed size.
  2. Most important point to note is when you change an element into this List corresponding element in original array will also be changed.
  3. Another important point is since List is fixed size, you can not add element into it. If you try you will get exception.

5) it doesn't copy the content of underlying array to create list

6)It returns a fixed size List not a read only List, although you can not add() or remove() elements on this List you can still change existing elements by using set method.  
  
  
  
Other way:

Arraylist newAssetList = new Arraylist();

newAssetList.addAll(Arrays.asList(asset));

When **ArrayList gets full** it creates another array and uses System.arrayCopy() to copy all elements from one array to another array. This is where insertion takes a lot of time.   
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Arraylist to Array

ArrayList assetTradingList = new ArrayList();

assetTradingList.add("Stocks trading");

assetTradingList.add("futures and option trading"); assetTradingList.add("electronic trading");

assetTradingList.add("forex trading");

assetTradingList.add("gold trading");

assetTradingList.add("fixed income bond trading");

String [] assetTradingArray = new String[assetTradingList.size()]; assetTradingList.toArray(assetTradingArray);

Linked List Implementation

<https://gist.github.com/es20641/1208340/4a0c8d59d63340118361a1ee64442edb769da43c>

Reverse a linked list – Refer - <http://www.java2blog.com/2014/07/how-to-reverse-linked-list-in-java.html>

There are two ways

* Iterative
* Recursive
* Iterative

// Reverse linkedlist using this function

public static Node reverseLinkedList(Node currentNode)

{

// For first node, previousNode will be null

Node previousNode=null;

  Node nextNode;

  while(currentNode!=null)

  {

   nextNode=currentNode.next;

  // reversing the link

   currentNode.next=previousNode;

  // moving currentNode and previousNode by 1 node

   previousNode=currentNode;

   currentNode=nextNode;

  }

  return previousNode;

}

* Recursive

**Base case:** Base case for this would be either node is null or node.next is null

|  |  |
| --- | --- |
|  | public static Node reverseLinkedList(Node node) {  if (node == null || node.next == null) {  return node;  }    Node remaining = reverseLinkedList(node.next);  node.next.next = node;  node.next = null;  return remaining;  } |

How can you protect a button by using spring security.?It should be shown for one person and should not be shown for another person.?

## 20. JSP Tag Libraries

Spring Security has its own taglib which provides basic support for accessing security information and applying security constraints in JSPs.

## 20.1 Declaring the Taglib

To use any of the tags, you must have the security taglib declared in your JSP:

<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

## 20.2 The authorize Tag

This tag is used to determine whether its contents should be evaluated or not. In Spring Security 3.0, it can be used in two ways [[25](http://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity-single.html" \l "ftn.d0e6888)]. The first approach uses a [web-security expression](http://docs.spring.io/spring-security/site/docs/3.1.x/reference/springsecurity-single.html#el-access-web), specified in the access attribute of the tag. The expression evaluation will be delegated to theSecurityExpressionHandler<FilterInvocation> defined in the application context (you should have web expressions enabled in your <http> namespace configuration to make sure this service is available). So, for example, you might have

<sec:authorize access="hasRole('supervisor')">

This content will only be visible to users who have

the "supervisor" authority in their list of <tt>GrantedAuthority</tt>s.

</sec:authorize>

A common requirement is to only show a particular link, if the user is actually allowed to click it. How can we determine in advance whether something will be allowed? This tag can also operate in an alternative mode which allows you to define a particular URL as an attribute. If the user is allowed to invoke that URL, then the tag body will be evaluated, otherwise it will be skipped. So you might have something like

<sec:authorize url="/admin">

This content will only be visible to users who are authorized to send requests to the "/admin" URL.

</sec:authorize>

To use this tag there must also be an instance of WebInvocationPrivilegeEvaluator in your application context. If you are using the namespace, one will automatically be registered. This is an instance of DefaultWebInvocationPrivilegeEvaluator, which creates a dummy web request for the supplied URL and invokes the security interceptor to see whether the request would succeed or fail. This allows you to delegate to the access-control setup you defined using intercept-url declarations within the <http> namespace configuration and saves having to duplicate the information (such as the required roles) within your JSPs. This approach can also be combined with a method attribute, supplying the HTTP method, for a more specific match.

The boolean result of evaluating the tag (whether it grants or denies access) can be stored in a page context scope variable by setting the var attribute to the variable name, avoiding the need for duplicating and re-evaluating the condition at other points in the page.

If i have 10 lacks records and i am reading all records in JDBC, Is this will give any exception or not and why?

Answer-http://makejavafaster.blogspot.in/2015/06/jdbc-fetch-size-performance.html

Assume we have table with 1 million rows and for any reason we have to fetch all records into JVM

|  |
| --- |
| PreparedStatement stmt = null; |
|  | ResultSet rs = null; |
|  |  |
|  | try { |
|  | stmt = conn. prepareStatement("select a, b, c from table"); |
|  | stmt.setFetchSize(200); |
|  |  |
|  | rs = stmt.executeQuery(); |
|  | while (rs.next()) { |
|  | ... |
|  | } |
|  | } |

Lets see what happens if fetchSize property is set to 10. When rs.next() is called for first time, the oracle driver fetches first 10 records from database and store them in a memory buffer. So, for next 9 calls to rs.next() records are retrieved from this buffer. After the buffer is fully read, subsequent rs.next() will force driver to fetch a new bunch of rows (10) into the buffer.

So if we want to read 10k rows with fetch size set to 10, the driver will make 1000 round trips to the database using the underlying connection. If we set the fetchSize to 500 the driver will perform only 20 round trips to our database.

Two important comments:

* fetchSize can be set on each Statement or PreparedStatement or even on ResultSet. By default, ResultSet uses fetchSize of Statement from which is born. The **default value** for Statement or PreparedStatement is jdbc **driver specific**
* fetchSize is **only a hint for the driver** – the Oracle driver respects this setting, while other drivers may ignore it and fetch all the records at once, for instance.

## Setting Fetch Size with Spring JdbcTemplate

When using Spring jdbc support you can do this in 2 ways:

### Ad hoc JdbcTemplate instance

|  |  |
| --- | --- |
|  | JdbcTemplate jdbc = new JdbcTemplate(dataSource); |
|  | jdbc.setFetchSize(200); |
|  |  |
|  | jdbc.query("select a, b, c from table", |
|  |  |
|  | new RowCallbackHandler() { |
|  | @Override |
|  | public void processRow(ResultSet rs) throws SQLException { |
|  | ... |
|  | } |
|  | } |
|  | ); |

**Shared JdbcTemplate instance**

|  |  |
| --- | --- |
|  | public class MyJdbcDaoImpl extends JdbcDaoSupport implements MyJdbcDao { |
|  |  |
|  | @Override |
|  | protected void initTemplateConfig() { |
|  | getJdbcTemplate().setFetchSize(200); |
|  | } |
|  |  |
|  | public MyResult loadAll() { |
|  | final MyResult result = new MyResult(); |
|  |  |
|  | getJdbcTemplate().query("select a, b, c from table", |
|  |  |
|  | new RowCallbackHandler() { |
|  | @Override |
|  | public void processRow(ResultSet rs) throws SQLException { |
|  | ... |
|  | result.add(...); |
|  | } |
|  | } |
|  | ); |
|  |  |
|  | } // end of loadAll |
|  | } |

When implementing a DAO that extends JdbcDaoSupport every call to getJdbcTemplate() returns the same shared JdbcTemplate instance. You can mix this with ad-hoc instances. For example, override initTemplateConfig()to set the default for this DAO but use ad-hoc JdbcTemplate for selected queries.

## Setting Fetch Size with Hibernate

With Hibernate you can specify jdbc fetchSize in two ways.

* First, you can set this as **global setting**.
* Second, you can overwrite this global setting for every **individual query** you want.

**Global setting**

* Hibernate gives you property called **hibernate.jdbc.fetch\_size** which you can set on your SessionFactory object

|  |
| --- |
| <bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean"> |
|  | ... |
|  | <property name="hibernateProperties"> |
|  | <props> |
|  | <prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop> |
|  | <prop key="hibernate.jdbc.fetch\_size">200</prop> |
|  | </props> |
|  | </property> |
|  | </bean> |

Individual query way –

**Fetch size per query**

Another method is to set fetchSize for particular query only. With Hibernate you can create HQL or SQL query in the following way:

|  |  |
| --- | --- |
|  | List list = session |
|  | .createQuery("from Person p where p.status = :status") |
|  | .setString("status", clientStatus) |
|  | .setFetchSize(200) |
|  | .list();  CONCLUSION- |

larger fetch size:

* usually gives you faster fetching
* requires more memory for buffers
* with some configurations consumes surprisingly much memory (oracle driver)

**The default fetchSize for the oracle driver is 10...**

Find the middle element in linked list –

Algo is

1. Use two pointer fastptr and slowptr and initialize both to head of linkedlist
2. Move fastptr by two nodes and slowptr by one node in each iteration.
3. When fastptr reaches end of nodes, the slowptr pointer will be  pointing to middle element.

public Node findMiddleNode(Node head)

{

Node slowPointer, fastPointer;

  slowPointer = fastPointer = head;

  while(fastPointer !=null) {

   fastPointer = fastPointer.next;

   if(fastPointer != null && fastPointer.next != null) {

    slowPointer = slowPointer.next;

    fastPointer = fastPointer.next;

   }

  }

  return slowPointer;

}

**To find nth element from last**

* Use two pointer firstPtr and secondPtr and initialize both to head of linkedlist
* Move firstPtr by n-1 nodes.
* Increment firstPtr and secondPtr until firstPtr.next not equal to null.
* SecondPtr will be at nth from end node.

public Node nthFromLastNode(Node head,int n)

{

  Node firstPtr=head;

  Node secondPtr=head;

  for (int i = 0; i < n; i++) {

   firstPtr=firstPtr.next;

  }

  while(firstPtr!=null)

  {

   firstPtr=firstPtr.next;

   secondPtr=secondPtr.next;

  }

  return secondPtr;

}

# How to detect loop in a linked list in java with example

* Use two pointer fastPtr and slowPtr and initialize both to head of linkedlist
* Move fastPtr by two nodes and slowPtr by one node in each iteration.
* If fastPtr and slowPtr meet at some iteration , then there is a loop in linkedlist.
* If fastPtr reaches to the end of linkedlist without meeting slow pointer then there is no loop in linkedlist (i.e fastPtr->next or fastPtr->next->next become null)

public boolean ifLoopExists() {

  Node fastPtr = head;

  Node slowPtr = head;

  while (fastPtr != null && fastPtr.next != null) {

   fastPtr = fastPtr.next.next;

   slowPtr = slowPtr.next;

   if (slowPtr == fastPtr)

    return true;

  }

  return false;

}

**Java Program to find second largest number in array:**

package org.arpit.java2blog;

public class FindSecondLargestMain {

public static void main(String args[])

{

int[] arr1={7,5,6,1,4,2};

int secondHighest=findSecondLargestNumberInTheArray(arr1);

System.out.println("Second largest element in the array : "+ secondHighest);

}

public static int findSecondLargestNumberInTheArray(int array[])

{

// Initialize these to the smallest value possible

int highest = Integer.MIN\_VALUE;

int secondHighest = Integer.MIN\_VALUE;

// Loop over the array

for (int i = 0; i < array.length; i++) {

// If current element is greater than highest

if (array[i] > highest) {

// assign second highest element to highest element

secondHighest = highest;

// highest element to current element

highest = array[i];

} else if (array[i] > secondHighest)

// Just replace the second highest

secondHighest = array[i];

}

// After exiting the loop, secondHighest now represents the second

// largest value in the array

return secondHighest;

}

}

**Merge Sort Algo**

**Steps to implement Merge Sort:**  
  
1) Divide the unsorted array into n partitions, each partition contains 1 element. Here the one element is considered as sorted.

2) Repeatedly merge partitioned units to produce new sublists until there is only 1 sublist remaining. This will be the sorted list at the end. -

**public** **class** **Mergesort** {

**private** **int[]** numbers;

**private** **int[]** helper;

**private** **int** number;

**public** **void** sort(**int[]** values) {

this.numbers = values;

number = values.length;

this.helper = **new** **int**[number];

mergesort(0, number - 1);

}

**private** **void** mergesort(**int** low, **int** high) {

*// check if low is smaller than high, if not then the array is sorted*

**if** (low < high) {

*// Get the index of the element which is in the middle*

**int** middle = low + (high - low) / 2;

*// Sort the left side of the array*

mergesort(low, middle);

*// Sort the right side of the array*

mergesort(middle + 1, high);

*// Combine them both*

merge(low, middle, high);

}

}

**private** **void** merge(**int** low, **int** middle, **int** high) {

*// Copy both parts into the helper array*

**for** (**int** i = low; i <= high; i++) {

helper[i] = numbers[i];

}

**int** i = low;

**int** j = middle + 1;

**int** k = low;

*// Copy the smallest values from either the left or the right side back*

*// to the original array*

**while** (i <= middle && j <= high) {

**if** (helper[i] <= helper[j]) {

numbers[k] = helper[i];

i++;

} **else** {

numbers[k] = helper[j];

j++;

}

k++;

}

*// Copy the rest of the left side of the array into the target array*

**while** (i <= middle) {

numbers[k] = helper[i];

k++;

i++;

}

}

}

**Complexity:**

**Best case:** O(nlogn) or O(n)  
**Average case:** O(nlogn)  
**Worst case:** O(nlogn)

**Bubble sort**

Bubble sort works by iterating first element to last element, comparing two adjacent elements and swapping them if they are not in correct order. Each iteration places next larger value to its correct place.

**class** BubbleSort {

**void** bubbleSort(**int** arr[])

    {

**int** n = arr.length;

**for** (**int** i = 0; i < n - 1; i++)

**for** (**int** j = 0; j < n - i - 1; j++)

**if** (arr[j] > arr[j + 1]) {

                    // swap arr[j+1] and arr[j]

**int** temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                }

    }

    /\* Prints the array \*/

**void** printArray(**int** arr[])

    {

**int** n = arr.length;

**for** (**int** i = 0; i < n; ++i)

            System.out.print(arr[i] + " ");

        System.out.println();

    }

    // Driver method to test above

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

    {

        BubbleSort ob = **new** BubbleSort();

**int** arr[] = { 5, 1, 4, 2, 8 };

        ob.bubbleSort(arr);

        System.out.println("Sorted array");

        ob.printArray(arr);

    }

}

**Complexity:**

The above function always runs **O(N2)** time even if the array is sorted. It can be optimized by stopping the algorithm if the inner loop didn’t cause any swap.

**static** **void** bubbleSort(**int** arr[], **int** n)

    {

**int** i, j, temp;

**boolean** swapped;

**for** (i = 0; i < n - 1; i++)

        {

            swapped = **false**;

**for** (j = 0; j < n - i - 1; j++)

            {

**if** (arr[j] > arr[j + 1])

                {

                    // swap arr[j] and arr[j+1]

                    temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp;

                    swapped = **true**;

                }

            }

            // IF no two elements were

            // swapped by inner loop, then break

**if** (swapped == **false**)

**break**;

        }

    }

***Worst and Average Case Time Complexity:****O(N2). The worst case occurs when an array is reverse sorted.****Best Case Time Complexity:****O(N). The best case occurs when an array is already sorted.*

**Quick Sort**

* Select an element from the array. This element is called the "pivot element". For example select the element in the middle of the array.
* All elements which are smaller than the pivot element are placed in one array and all elements which are larger are placed in another array.
* Sort both arrays by recursively applying Quicksort to them.
* Combine the arrays.

**public** **class** **Quicksort** {

**private** **int[]** numbers;

**private** **int** number;

**public** **void** sort(**int[]** values) {

*// check for empty or null array*

**if** (values ==null || values.length==0){

**return**;

}

this.numbers = values;

number = values.length;

quicksort(0, number - 1);

}

**private** **void** quicksort(**int** low, **int** high) {

**int** i = low, j = high;

*// Get the pivot element from the middle of the list*

**int** pivot = numbers[low + (high-low)/2];

*// Divide into two lists*

**while** (i <= j) {

*// If the current value from the left list is smaller than the pivot*

*// element then get the next element from the left list*

**while** (numbers[i] < pivot) {

i++;

}

*// If the current value from the right list is larger than the pivot*

*// element then get the next element from the right list*

**while** (numbers[j] > pivot) {

j--;

}

*// If we have found a value in the left list which is larger than*

*// the pivot element and if we have found a value in the right list*

*// which is smaller than the pivot element then we exchange the*

*// values.*

*// As we are done we can increase i and j*

**if** (i <= j) {

exchange(i, j);

i++;

j--;

}

}

*// Recursion*

**if** (low < j)

quicksort(low, j);

**if** (i < high)

quicksort(i, high);

}

**private** **void** exchange(**int** i, **int** j) {

**int** temp = numbers[i];

numbers[i] = numbers[j];

numbers[j] = temp;

}

}

time complexity of **O(N\*logN)**.

**Binary Search**

public  static int binarySearch(int[] sortedArray, int elementToBeSearched) {

  int first = 0;

        int last = sortedArray.length - 1;

      while (first < last) {

          int mid = (first + last) / 2;  // Compute mid point.

          if (elementToBeSearched < sortedArray[mid]) {

           last = mid;     // repeat search in first half.

          } else if (elementToBeSearched > sortedArray[mid]) {

              first = mid + 1;  // Repeat sortedArray in last half.

          } else {

              return mid;     // Found it. return position

          }

      }

      return -1;    // Failed to find element

  }

**Logic for prime number**

#### Program logic:

If any number which is not divisible by any other number which is less than or equal to square root of that number, then it is prime number.

public static boolean isPrime(int number)

{

  for (int i = 2; i <=Math.sqrt(number); i++) {

   if(number%i==0)

    return false;

  }

  return true;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|  |
| --- |
| What will be the output of the program?  int i = (int) Math.random(); |
| |  |  | | --- | --- | | [**A.**](javascript:%20void%200;) | i = 0 | | [**B.**](javascript:%20void%200;) | i = 1 | | [**C.**](javascript:%20void%200;) | value of i is undetermined | | [**D.**](javascript:%20void%200;) | Statement causes a compile error |   **Answer:** Option **A**  **Explanation:**  Math.random() returns a double value greater than or equal to 0 and less than 1. Its value is stored to an int but as this is a narrowing conversion, a cast is needed to tell the compiler that you are aware that there may be a loss of precision.  The value after the decimal point is lost when you cast a double to int and you are left with 0. |

|  |
| --- |
| What will be the output of the program?  int i = 1, j = 10;  do  {  if(i++ > --j) /\* Line 4 \*/  {  continue;  }  } while (i < 5);  System.out.println("i = " + i + "and j = " + j); /\* Line 9 \*/ |
| |  |  | | --- | --- | | [**A.**](javascript:%20void%200;) | i = 6 and j = 5 | | [**B.**](javascript:%20void%200;) | i = 5 and j = 5 | | [**C.**](javascript:%20void%200;) | i = 6 and j = 6 | | [**D.**](javascript:%20void%200;) | i = 5 and j = 6 |   **Answer:** Option **D**  **Explanation:**  This question is not testing your knowledge of the continue statement. It is testing your knowledge of the order of evaluation of operands. Basically the prefix and postfix unary operators have a higher order of evaluation than the relational operators. So on line 4 the variable i is incremented and the variable j is decremented before the greater than comparison is made. As the loop executes the comparison on line 4 will be:  if(i > j)  if(2 > 9)  if(3 > 8)  if(4 > 7)  if(5 > 6) at this point i is not less than 5, therefore the loop terminates and line 9 outputs the values of i and j as 5 and 6 respectively.  The continue statement never gets to execute because i never reaches a value that is greater than j. |

Calling stored procedure in hibernate

## 1. Native SQL – createSQLQuery

You can use **createSQLQuery()** to call a store procedure directly.

Query query = session.createSQLQuery(

"CALL GetStocks(:stockCode)")

.addEntity(Stock.class)

.setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

Stock stock = (Stock)result.get(i);

System.out.println(stock.getStockCode());

}

## 2. NamedNativeQuery in annotation

Declare your store procedure inside the **@NamedNativeQueries** annotation.

//Stock.java

...

@NamedNativeQueries({

@NamedNativeQuery(

name = "callStockStoreProcedure",

query = "CALL GetStocks(:stockCode)",

resultClass = Stock.class

)

})

@Entity

@Table(name = "stock")

public class Stock implements java.io.Serializable {

...

Call it with **getNamedQuery()**.

Query query = session.getNamedQuery("callStockStoreProcedure")

.setParameter("stockCode", "7277");

List result = query.list();

for(int i=0; i<result.size(); i++){

Stock stock = (Stock)result.get(i);

System.out.println(stock.getStockCode());

}