**2014 Java Revisted**

**Difference between Connected vs Disconnected RowSet in Java JDBC | RowSetProvider, RowSetFactory Example**RowSet interface is another key interface from JDBC API, which extends the ResultSet interface to provide support for the JavaBean component model. Along with convenient methods to set connection parameters e.g. setUrl(), setUsername(), setPassword() and setCommand(), it also defines getters and setters for different data types e.g. getLong() andsetLong(). Another feature of RowSet is that it supports event listeners. You can attach listeners to RowSet object to listen for events, much like [Observer design pattern](http://javarevisited.blogspot.sg/2011/12/observer-design-pattern-java-example.html). RowSet acts as Subject and notifies Observer on events like change in cursor location, change in row or change in the entire RowSet. There are two types of RowSet available in JDBC 4.1 API, connected and disconnected. A **connected RowSet** always keeps connection with database, while a **disconnected RowSet** connects to database, get the data and then close the connection. This doesn't prevent from working with data though, you can still change data in disconnected state, but to commit, connection needs to be re-established by RowSet. Java provides five different types ofRowSet, JdbcRowSet, JoinRowSet, CachedRowSet, FilteredRowSet and WebRowSet. Out of these five *only JdbcRowSet is connected RowSet,* rest of them are disconnected row sets. It's very important to understand these new concepts form JDBC API, there are very [popular JDBC Interview questions](http://javarevisited.blogspot.sg/2012/12/top-10-jdbc-interview-questions-answers.html) based upon these concepts e.g. *difference between RowSet and ResultSet*and *difference between Connected vs Disconnected RowSet*. Once you are familiar with key concept of RowSet class, you should be able to answer these question by your own.  
**Difference between Connected and Disconnected RowSet**

As I mentioned, main difference between connected and disconnected RowSet is that former always keeps a database connection, while later doesn't. It connects to database, get the data and then close the connection. Here are couple of more differences between them :  
  
1) Disconnected database connected to the database only when they want to read or write, all the times they are disconnected from the [database](http://javarevisited.blogspot.sg/2011/11/database-transaction-tutorial-example.html), on the other hand, Connected database keep JDBC connection alive all time.  
  
2) Only JdbcRowSet from JDBC 4.1 API is connected, rest like CachedRowSet, WebRowSet are disconnected RowSet implementation.  
  
3) Disconnected RowSet are [Serializable](http://java67.blogspot.sg/2012/08/what-is-transient-variable-in-java.html)and that's why suitable for sending over the network  
**RowSet Example in Java**

Let's see a simple example of RowSet in Java. In this example, we will retrieve employees record using RowSet instead of ResultSet interface. So far, you might have only seen a ResultSet object returning query result, but from JDBC 4.1 API, you can use any implementation of RowSet implementation for querying database and retrieving result. In this JDBC example, we will use JdbcRowSet class. In order to create object of JdbcRowSet, you need a RowSetFactory, which can be created by using newFactory() method of RowSetProvider. Once you have an object of RowSet, you can pass configuration details like database URL, username and password by calling their respective setProperties() method e.g. setURL(), setUsername()and setPassword(). After configuration, its time to specify our [SQL SELECT Query](http://javarevisited.blogspot.sg/2011/10/selct-command-sql-query-example.html), we will use method called setCommand() for providing query. Now you can execute the query by calling execute() method. Did you notice a much cleaner and simple API? After query execution its time to retrieve data, which is very similar to what you are used by calling various getXXX()method from ResultSet. You just need to iterate through RowSet by checking hasNext() method in while loop and getting data row by row.

import javax.sql.rowset.\*;

import java.sql.\*;

/\*\*

\* Java Program to illustrate how to use RowSet, RowSetProvider, and RowSetFactory in JDBC

\* In this example, RowSet is used to retrieve data from database instead of ResultSet.

\* @author Javin Paul

\*/

public class RowSetDemo {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/addressBook";

String userName = "root";

String password = "mysql123";

try {

// first, create a factory object for rowset

RowSetFactory rowSetFactory = RowSetProvider.newFactory();

// create a JDBC rowset from the factory

JdbcRowSet rowSet = rowSetFactory.createJdbcRowSet();

// Set connection properties

rowSet.setUrl(url);

rowSet.setUsername(userName);

rowSet.setPassword(password);

// Set SQL Query to execute

rowSet.setCommand("SELECT \* FROM contact");

rowSet.execute();

System.out.println("id \tName \tDepartment \tEmail \t\Salary");

// Iterating over RowSet

while (rowSet.next()) {

System.out.println(rowSet.getInt("id") + "\t"

+ rowSet.getString("name") + "\t"

+ rowSet.getString("department") + "\t"

+ rowSet.getString("email") + "\t"

+ rowSet.getString("salary"));

}

} catch (SQLException sqle) {

sqle.printStackTrace();

}

}

}

**Output**

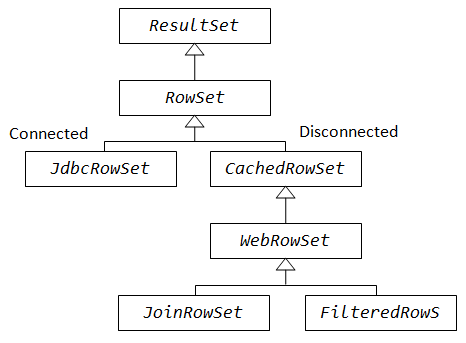
Id Name Department Email Salary

1 Jack Sales jack@bluechip.com 9000

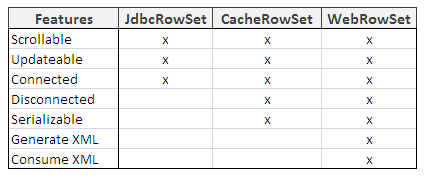
2 Jill Marketing jill@bluechip.com 8000

3 Tom Accounts tom@bluechip.com 7000

You can see how easy is to use RowSet in DAO  of your Java program, or you can also use them along with [DAO Design Pattern](http://javarevisited.blogspot.sg/2013/01/data-access-object-dao-design-pattern-java-tutorial-example.html). Since RowSet inherit from java.sql.ResultSet interface, you can actually pass RowSet implementation to any method, which expect ResultSet. This diagram will give you more idea about ResultSet and RowSet class hierarchy.  It also useful to find out which RowSet implementation are Connected and which one are Disconnected. You can see that there are two major implementation of RowSet interface, JdbcRowSet which is connected and CachedRowSet which is disconnected.

[](http://1.bp.blogspot.com/-ecgVBHt5fz4/U15am0Kn2gI/AAAAAAAABdA/tNrW3AQ4FDc/s1600/RowSet+Classes+in+JDBC+Java.png)

CachedRowSet further has couple of more implementation in terms of WebRowSet and its subclasses. This table provides a nice comparison of features supported by different RowSet implementation in JDBC API.

[](http://4.bp.blogspot.com/-X5CtG0KwhWw/U15ammfpkwI/AAAAAAAABdE/XKuhB1JZ-dU/s1600/JdbcRowSet+Connected+vs+Disconnected+Feature+Comparison.png)

You can see that JdbcRowSet is Scrollable, Updateable and Connected, while CachedRowSet is also [Serializable](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html) because its disconnected. WebRowSet further adds feature of generating and consuming XML.  
  
**Things to remember about RowSet, ResultSet and RowSetFactory**

It's time to revise few key points about these classes from JDBC 4.1 API.  
  
1) RowSet extends ResultSet interface to add support of Java Bean model e.g. it adds methods like setUrl() which makes it easy to connect to database.  
  
2) There are two types of RowSet, connected and disconnected. connected RowSet object make a connection with the database and maintain that connection throughout its life cycle. On the other hand, a disconnected RowSet makes a connection with [data source](http://javarevisited.blogspot.sg/2012/06/jdbc-database-connection-pool-in-spring.html), read data from it and close the connection. A disconnected RowSet can still make changes to its data while it is not connected with database, but it must re-establish connection to commit those changes.  
  
3) There are five implementations of RowSet available in JDBC 4.1 API, JdbcRowSet, CachedRowSet, JoinRowSet, WebRowSet and FilteredRowSet. Out of these five, only JdbcRowSet is a connected RowSet, rest of them are disconnected RowSets.  
  
4) RowSetProvider provides methods to get a RowSetFactory implementation, which can be used to instantiate a relevant RowSet implementation. Both of these are added from Java 1.7 only.  
  
5) Disconnected RowSet classes are also Serializable in Java.  
  
That's all about **What is RowSet in JDBC**, How to use RowSet to connect to database, and *difference between Connected and Disconnected RowSet in Java*. We have also see example of using RowSetProvider and RowSetFactory which can be used to  create different types of RowSet in Java. Remember, only JdbcRowSet is the connected RowSet, rest of them are disconnected one.  
**Law of Demeter in Java - Principle of least Knowledge - Real life Example**Law of Demeter also known as*principle of least knowledge* is a coding principle, which says that a module should not know about the inner details of the objects it manipulates. If a code depends upon internal details of a particular object, there is good chance that it will break as soon as internal of that object changes. Since [Encapsulation](http://javarevisited.blogspot.com/2012/03/what-is-encapsulation-in-java-and-oops.html) is all about hiding internal details of object and exposing only operations, it also assert *Law of  Demeter*. One mistake many Java programmer makes it exposing internal detail of object using getter methods and this is where principle of least knowledge alerts you. I first come to know about this principle, while reading one of the [must read programming book](http://javarevisited.blogspot.com/2013/05/which-programming-book-would-you-buy-if.html),  Robert C. Martin's Clean code. Apart from many good thing the book teaches you, "principle of least knowledge" is one principle, which I still remember. Like many bad things, you will tempt to violate Law of Demeter, because of beautiful chaining of methods written in fluent style. On surface it looks pretty good, but as soon as you think about principle of least knowledge, you start seeing the real picture. In this article, we will see formal definition of Law of Demeter and explore code snippet which violates this principle.  
  
Law of Demeter

According to Law of Demeter, a method M of object O should only call following types of methods :

Methods of Object O itself

1. Methods of Object passed as an argument
2. Method of object, which is held in instance variable
3. Any Object which is created locally in method M

More importantly method should not invoke methods on objects that are returned by any subsequent method calls specified above and as Clean Code says "talk to friends, not to strangers". Apart from knowing object oriented programming basic concepts e.g. [Abstraction](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html), [Polymorphism](http://javarevisited.blogspot.sg/2011/08/what-is-polymorphism-in-java-example.html), [Inheritance](http://java67.blogspot.com/2012/08/what-is-inheritance-in-java-oops-programming-example.html)and [SOLID design principle](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html), it's also worth knowing useful principle like this, which has found it's way via experience. In following example, we will see how a method can violate above rules to violate Law of Delimiter.  
public class LawOfDelimterDemo {

    /\*\*

     \* This method shows two violations of "Law of Delimiter" or "Principle of least knowledge".

     \*/

    public void process(Order o) {

        // as per rule 1, this method invocation is fine, because o is a argument of process() method

        Message msg = o.getMessage();

        // this method call is a violation, as we are using msg, which we got from Order.

        // We should ask order to normalize message, e.g. "o.normalizeMessage();"

        msg.normalize();

        // this is also a violation, instead using temporary variable it uses method chain.

        o.getMessage().normalize();

        // this is OK, a constructor call, not a method call.

        Instrument symbol = new Instrument();

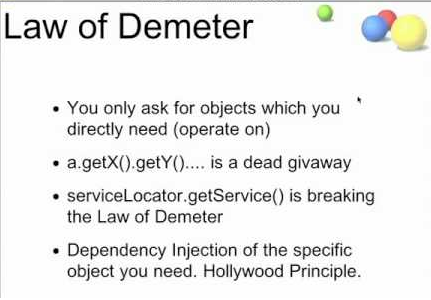
        // as per rule 4, this method call is OK, because instance of Instrument is created locally.

        symbol.populate();

    }

}

You can see that when we get internal of Order class and  call a method on that object, we violate *Law of delimiter*, because now this method knows about Message class. On the other hand calling method on Order object is fine because its passed to the method as parameter.  This image nicely explains what you need to do to follow Law of Demeter.

[](http://2.bp.blogspot.com/-eyIv7MV4CDw/U4XkaYxuyuI/AAAAAAAABj4/RbNXifaTPaE/s1600/Law+of+Demeter+in+Java+with+Example.jpg)

Let's see another example of code, which violates the Law of Demeter and how does it affect code quality.  
public class XMLUtils {

public Country getFirstBookCategoryFromXML(XMLMessage xml) {

return xml.getXML().getBooks().getBookArrary(0).getBookHeader().getBookCategory();

}

}

This code is now dependent upon lot of classes e.g.  
    XMLMessage  
    XML  
    Book  
    BookHeader  
    BookCategory  
  
Which means this function knows about XMLMessage, XML, Book, BookHeader and BookCategory. It knows that XML has list of Book, which in-turn has BookHeader and which internally has BookCategory, that's a lot of information. If any of the intermediate class or accessor method in this chained method call changes, then this code will break. This code is highly coupled and brittle. It's much better to put the responsibility of finding internal data into the object, which owns it. If we look closely, we should only call getXML()method because its method from XMLMessage class, which is passed to method as argument.  Instead of putting all this code in XMLUtils, should be putting on BookUtils or something similar, which can still follow Law of Demeter and can return the required information.  
  
That's all on this **example of Law of Demeter or "Principle of least knowledge"**. It's better not to have a chain of methods, originating from unknown object, which may change. I also suggest to read Chapter 6 of Clean Code, a must read book for any software developers.  
  
**Why use SerialVersionUID inside Serializable class in Java**  
Serialization and SerialVersionUID is always remains a puzzle for many Java developers. I often see questions like what is this SerialVersionUID, or what will happen if I don't declare SerialVersionUID in my Serializable class? Apart from complexity involved and rare use, one more reason of these questions is Eclipse IDE's warning against absence of SerialVersionUID e.g. *"The Serializable class Customer does not declare a static final SerialVersionUID field of type long"*. In this article, you will not only learn basics of Java SerialVersionUID but also it's effect during serialization and de-serialization process. When you declare a class as Serializable by implementing marker interface java.io.Serializable, Java runtime persist instance of that class into disk by using default Serialization mechanism, provided you have not customized the process using [Externalizable interface](http://javarevisited.blogspot.com/2012/01/serializable-externalizable-in-java.html). During serialization, Java runtime creates a version number for a class, so that it can de-serialize it later. This version number is known as *SerialVersionUID*in Java. If during de-serialization, SerialVersionUID doesn't match than process will fail withInvalidClassException as  *Exception in thread "main" java.io.InvalidClassException*, also printing class-name and respective SerialVersionUID. Quick solution to fix this problem is copying SerialVersionUID and declaring them as private static final long constant in your class. In this article, we will learn about why we should use SerialVersionUID in Java and How to use *serialver*JDK tool for generating this ID. If you are new to serialization, you can also see [Top 10 Java Serialization Interview question](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html) to gauge your knowledge and find gap in your understanding for further reading. Similar to Concurrency and Multi-threading, Serialization is another topic, which deserve couple of reading.  
**Why use SerialVersionUID in Java**

As I said, when we don't declare SerialVersionUID, as a static, final and long value in our class, Serialization mechanism creates it for us. This mechanism is sensitive to many details including fields in your class, there access modifier, the interface they implement and even different Compiler implementations, any change on class or using different compiler may result in different *SerialVersionUID*, which many eventually stop reloading serialized data. It's too risky to rely on Java Serialization mechanism for generating this id, and that's why it's recommended to declare explicit SerialVersionUID in your Serializable class. I strongly suggest to read Joshua Bloch's classic Java title, [Effective Java](http://www.amazon.com/Java-SE8-Really-Impatient-Course/dp/0321927761?tag=javamysqlanta-20) to understand Java Serialization and issues of incorrect handling it. By the way JDK also provides a tool called serialver, located in bin directory of [JAVA\_HOME](http://javarevisited.blogspot.sg/2012/02/how-to-set-javahome-environment-in.html) folder, in my machine C:\Program Files\Java\jdk1.6.0\_26\bin\serialver.exe, which can be used to generate SerialVersionUID for old classes. This is very helpful, in case you have made changes in your class, which is breaking Serialization and your application is not able toreload serialized instances. You can simply use this tool to create SerialVersionUID for old instances and then use it explicitly in your class by declaring a [private](http://javarevisited.blogspot.com/2012/03/private-in-java-why-should-you-always.html), [static](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), [final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html)and [long](http://javarevisited.blogspot.sg/2012/12/how-to-convert-string-to-long-in-java-4-examples.html) SerialVersionUID field. By the way, it's highly recommend, both due to performance and security reason to use customized binary format for Serialization, once again [Effective Java](http://www.amazon.com/Java-SE8-Really-Impatient-Course/dp/0321927761?tag=javamysqlanta-20) has couple of Items, which explains benefits of custom format in great details.  
**How to use serialver JDK tool to generate SerialVersionUID**

You can use JDK's **serialver**tool to generate SerialVersionUID for classes. This is particularly useful for evolving classes, it returns SerialVersionUID in format easy to copy. You can use serialver JDK tool as shown in below example :

$ serialver

**use:** serialver [-classpath classpath] [-show] [classname...]

$ serialver -classpath . Hello

Class Hello is not Serializable.

$ serialver -classpath . Hello

Hello: static final long SerialVersionUID = -4862926644813433707L;

You can even use serialver tool in GUI form by running command  $ serialver -show, this will open the serial version inspector, which takes full class name and shows it's Serial version.  
**Summary**

Now we know what is SerialVersionUID and why it's important to declare it in Serializable class, it's time to revise some of the important fact, related to Java SerialVersionUID.  
  
1) SerialVersionUID is used to version serialized data. You can only de-serialize a class if it's SerialVersionUID matches with the serialized instance.  
  
2) When we don't declare SerialVersionUID in our class, Java runtime generates it for us, but that process is sensitive to many class meta data including number of fields, type of fields, access modifier of fields, interface implemented by class etc. You can find accurate information in Serialization documentation from Oracle.  
3) It's recommended to declare SerialVersionUID as private static final long variable to avoid default mechanism. Some IDE like [Eclipse](http://javarevisited.blogspot.com/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) also display warning if you miss it e.g. *"The Serializable class Customer does not declare a static final SerialVersionUID field of type long"*. Though you can disable this warnings by going to Window > Preferences > Java > Compiler > Errors / Warnings > Potential Programming Problems, I suggest not to do that. Only case, I see being careless is when restoring of data is not needed. Here is how this error looks like in Eclipse IDE, all you need to do is accept first quick fix.  
4) You can even use serialver tool from JDK to generate Serial Version for classes in Java. It also has a GUI, which can be enable by passing -show parameter.  
  
5) It's Serialization best practice in Java to explicitly declare SerialVersionUID, to avoid any issues during de-serialization especially if you are running a client server application which relies on serialized data e.g. RMI.  
  
That's all about **SerialVersionUID in Java**. Now we know that Why it's important to declare SerialVersionUID right into the class. You can thanks your IDE for this reminder, which may potentially break de-serialization of your class.  
  
**How String in Switch works in Java 7**Ever Since Java allows using String variable in switch and case statements, there are many programmers using this feature in code, which can be better written using integer and enum pattern. This was one of the [popular features of JDK 7 release](http://javarevisited.blogspot.sg/2014/04/10-jdk-7-features-to-revisit-before-you.html), including automatic resource management and multi exception catch blocks. Though I upfront didn't like this feature because of better alternative available in terms of using enumeration type, I am not totally against this feature. One reason of this is convenience and given usage of String in Java program, it's quite handy as well, but I prefer to learn more before using any new feature in [production code](http://javarevisited.blogspot.sg/2011/09/how-to-write-production-quality-code.html). When I first come to know about this feature, I had an idea that String in Switch can be implemented using equals() and hashCode() method, I was more interested on *how String in Switch works in Java 7*. One more reason I was curious to know about internal working of this feature because I wanted to ask this during Java interviews, having one of such question makes interviews little more interesting. Testing was simple, you just need to write code using String variables in switch block, and then decompile the code to see, how compiler has translated them. So what are we waiting for, Let's see *how String in switch block actually works?*  
**Original Code :**

This is the simple test program, which has a [main method](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html) and a switch block, which is operating on String variable. String argument is provided at the time of running this program, which is then accessed from String array argument of main method. We have three modes to start our application, Active, Passive and Safe. Though its better to use Enum to represent such kind of well known fixed values, if you decided to use String, make sure you write it down in capital case to avoid case-sensitive issue with lower case and camel case. You can also see [this](http://javarevisited.blogspot.sg/2011/08/string-switch-case-jdk7-example.html) tutorial to learn more about correctly using String in switch expressions in Java SE 7.  
/\*\*

\* Java Program to demonstrate how string in switch functionality is implemented in

\* Java SE 7 release.

\*/

public class StringInSwitchCase{

public static void main(String[] args) {

String mode = args[0];

switch (mode) {

case "ACTIVE":

System.out.println("Application is running on Active mode");

break;

case "PASSIVE":

System.out.println("Application is running on Passive mode");

break;

case "SAFE":

System.out.println("Application is running on Safe mode");

}

}

}

You need to install JDK 1.7 to compile and run this code. You can use any version of JDK 7.  
**Decompiled Code :**

This is the decompiled version of above class after being compiled on jdk1.7.0\_40 version. If you are new in Java and want to learn [how to decompile Java class file](http://javarevisited.blogspot.sg/2013/01/how-to-decompile-class-file-in-java-eclipse-javap-example.html) for reverse engineering, see that post. Since with every new release we are getting more and more syntactic sugar, knowing how to decompile a class has become very important for all level of Java programmers. The gap between code you wrote and what get executed is widening very fast.  Basic knowledge of Java class file format and byte code instruction will only going to help your.  Java 8 recently released a key feature called [lambda expression](http://javarevisited.blogspot.sg/2014/02/10-example-of-lambda-expressions-in-java8.html) also takes helps of compiler to implement Anonymous class internally, you can decompile your class file to see methods added by compiler.  
/\*\*

\* Reverse Engineered code to show how String in Switch works in Java.

\*/

import java.io.PrintStream;

public class StringInSwitchCase{

public StringInSwitchCase() {

}

public static void main(string args[]) {

String mode = args[0];

String s;

switch ((s = mode).hashCode()) {

default:

break;

case -74056953:

if (s.equals("PASSIVE")) {

System.out.println("Application is running on Passive mode");

}

break;

case 2537357:

if (s.equals("SAFE")) {

System.out.println("Application is running on Safe mode");

}

break;

case 1925346054:

if (s.equals("ACTIVE")) {

System.out.println("Application is running on Active mode");

}

break;

}

}

}

If you at this code, you will find out that String in Switch works by using [hashCode() and equals() method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). Remember, we can only use integer variable in switch case i.e. variable of type byte, short, char, and int. Good thing is return type of hashCode() method is int, not long. By the way this one way to remember this fact as well, which I often forget/get confused by myself. If you look closely, switch is on hash code and then a safety check by comparing String with [equals() method](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html), this check is required because two unequal object can have same hash code. So performance wise, it is  not as fast as using [enum constants on switch case](http://java67.blogspot.sg/2012/09/how-to-use-java-enum-in-switch-case-example.html) or using pure integer constant on switch, but its not too bad at all. Since Java compiler is only using one additional method equals(), which can be very fast if you are comparing String literals i.e. when "abc" == "abc". If you are also considering about calling to hashCode() method, yes that is another 1 time additional cost, because once created, String cache there hash code, as discussed on my favourite article [why String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html). So cost of calling hashCode, will not be significant if this switch case is used in a tight loop e.g. loop to process items or game engine loops to render screens. Nevertheless, I still consider using String in switch statement and using it represent fixed number of things is not a good practice, Enumeration type in Java is there for a reason, and every Java programmer must use it.  
  
  
That's all on **How String in Switch works in Java 7**. As expected it uses hashCode() method for switching and equals() method for verification, This means it's just a syntactic sugar, rather than an inbuilt native functionality. Now choice is yours, I am personally not a big fan of *using String in Switch case* as it result in brittle code, case-sensitive issue, and no compile time check for invalid input. In fact plain old integer constants are my favourite for performance critical code and Enumeration type in Java, where [readability](http://javarevisited.blogspot.sg/2012/11/What-is-timeunit-sleep-over-threadsleep.html) and [code quality](http://javarevisited.blogspot.sg/2014/02/why-static-code-analysis-is-important.html) is more important. In fact in 99.99% cases enum is better choice then String or integer variable, its the very same reason they exists in Java programming language. All this feature has done is promoted this bad coding practices, I struggle hard to find a proper use case of using String in switch cases with a set of inputs in any other purpose then testing and debugging, let me know if you have a convincing reason of using String in switch case in your project, may be that will change my mind.  
  
**How to create Custom Exception in Java - Tutorial Example**  
Sometimes we need to create custom Exception in Java, i.e. Exceptions which are not defined in JDK or any third party library your application is using. Though it’s widely recommended on several [Exception best practices](http://javarevisited.blogspot.sg/2013/03/0-exception-handling-best-practices-in-Java-Programming.html) article, even Joshua Bloch has recommended in [Effective Java](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) to prefer standard exception over custom exception, sometimes you really need it. There are certain guidelines to help to find whether you really need a custom exception or not. You should write your own exception classes if you answer yes to any of the following questions; otherwise, you can probably use someone else's.

* Do you need an exception type that isn't represented by those in the Java platform?
* Would it help users if they could differentiate your exceptions from those thrown by classes written by other vendors?
* Does your code throw more than one related exception?
* If you use someone else's exceptions, will users have access to those exceptions? The similar question is, should your package be independent and self-contained?

## Custom Exception or Custom Message with Standard Exception?

For example if you declare an Exception that doesn't provide any useful information other than a custom name then it probably uses generic Exception class with a custom message as shown in below example:

public class DuplicateIDException extends *Exception* {}

This custom exception doesn't provide any extra information e.g. alternative ids, and that's why can be easily replaced by a custom message and standard Exception class, as shown below:

throw new Exception("ID already taken");

Even better, if you think the client code is not going to take any action other than logging if the id is already taken, throw an unchecked exception:  
  
throw new RuntimeException("ID already taken");

Checked or Unchecked?

Once you make the decision to create custom Exception, the next thing is to decide on [checked vs unchecked exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html). As I said before, by default make your exception unchecked and you will find it whether it should be checked while writing client code. General guideline is to make an exception unchecked if the client code is not going to take any action other than logging.

How to create Custom Exception in Java - The Code  
  
Here is our complete code example of creating custom or user defined Exception in Java.  In our example, we have created NoSuchProductException, which is thrown by methods returning products. This is an unchecked Exception as we made it inherit from RuntimeException. It inherit getMessage() method from Throwable and also has a method getProductId()which returns product id for which this exception has caused. Don't make an Exception class as [nested class](http://javarevisited.blogspot.sg/2012/12/inner-class-and-nested-static-class-in-java-difference.html) even if it's used only by one class, always declare Exceptions in their own class.  
import java.util.HashMap;

import java.util.Map;

/\*\*

\* Java Program to create custom exception and examples to show how to use

\* custom exception in Java.

\*

\* @author Javin Paul

\*/

public class CustomExceptionDemo {

private static final Map&lt;Integer, String&gt; products = new HashMap<>();

static {

products.put(100, "Coke");

products.put(101, "KitKat");

products.put(102, "Bisuits");

products.put(103, "Toast");

}

public static void main(String args[]) {

CustomExceptionDemo t = new CustomExceptionDemo();

t.getProduct(1000);

}

public String getProduct(int id) {

if (products.get(id) == null) {

throw new NoSuchProductException("No such product exists", id);

}

return products.get(id);

}

}

class NoSuchProductException extends *RuntimeException* {

private int productId;

public NoSuchProductException() {

super();

}

public NoSuchProductException(String message, int productId) {

super(message);

this.productId = productId;

}

public NoSuchProductException(String message, int productId, Throwable cause) {

super(message, cause);

this.productId = productId;

}

@Override

public String toString() {

return super.toString();

}

@Override

public String getMessage() {

return super.getMessage() + " for productId :" + productId;

}

public int getProductId() {

return productId;

}

}

Output:

Exception in thread "main" NoSuchProductException: No such product exists for productId :1000

at CustomExceptionDemo.getProduct(CustomExceptionDemo.java:26)

at CustomExceptionDemo.main(CustomExceptionDemo.java:21)

**Things to remember while creating Custom Exception in Java**

Though creating a custom, exception is as easy as subclassing java.lang.Exception class, there are few best practices you can follow to make most of it. There is so much criticism of checked exception due to boilerplate require to handle it, you will hardly create your custom exception as checked.  
  
1) Don’t' use Exception to control application behavior. Exception handling is very expensive as it requires native calls to copy [stack trace](http://javarevisited.blogspot.sg/2013/04/how-to-get-current-stack-trace-in-java-thread.html), each time exception is created.  
  
2) While creating a custom exception, prefer to create an unchecked, Runtime exception than a checked exception, especially if you know that client is not going to take any reactive action other than logging.  
  
3) If your custom exception is created by passing another exception, then always contain original Exception as a source; use constructor which takes Exception rather than only message String.  
  
4) Apart from providing default no argument constructor on your custom Exception class, consider providing at least two more constructors, one which should accept a failure message and other which can accept another Throwable as the cause.  
  
5) If possible, avoid creating custom Exception and re-use existing, standard Exception classes from JDK itself. Most of the time you will realize that all you need is a form of IllegalArgumentException or ParseException or something similar.  
  
6) While defining custom Exception, one of the most common mistake programmer make is to think that constructor is inherited from java.lang.Exception class, for example, they think that their Exception class will automatically inherit default no argument constructor and the one which takes a String message. This is not true. [The constructor is not inherited in Java](http://javarevisited.blogspot.sg/2012/12/what-is-constructor-in-java-example-chainning-overloading.html), not even [default constructor](http://javarevisited.blogspot.sg/2014/01/why-default-or-no-argument-constructor-java-class.html). It's actually added by the compiler rather than inherited from parent class. That's why I have declared two constructors, one with String parameter and other as Throwable parameter:

public NoSuchProductException(String message, int productId) {

super(message);

this.productId = productId;

}

public NoSuchProductException(String message, int productId, Throwable cause) {

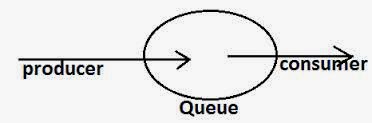
super(message, cause);

this.productId = productId;

}

This is actually the standard way of creating custom Exception in Java. In order to save time, you can even create a template of above class in Eclipse IDE.  
  
7) For readable code, it's good practice to append the string Exception to the names of all classes that inherits (directly or indirectly) from the Exception class e.g. instead of naming your class IncorrectPassword, name it IncorrectPasswordException.  
  
**SynchronousQueue Example in Java - Produer Consumer Solution**  
SynchronousQueue is special kind of BlockingQueue in which each insert operation must wait for a corresponding remove operation by another thread, and vice versa. When you call put() method on SynchronousQueue it blocks until another thread is there to take that element out of the Queue. Similarly, if a thread tries to remove an element and no element is currently present, that thread is blocked until another thread puts an element into the queue. You can correlated SynchronousQueue with athletes ([threads](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html)) running with Olympic torch, they run with torch (object need to be passed) and passes it to other athlete waiting at other end. If you pay attention to the name, you will also understand that it is named SynchronousQueue with a reason, it *passes data synchronously to other thread;* it wait for the other party to take the data instead of just putting data and returning (asynchronous operation). If you are familiar with CSP and Ada, then you know that synchronous queues are similar to rendezvous channels. They are well suited for hand-off designs, in which an object running in one thread must sync up with an object running in another thread in order to hand it some information, event, or task. In earlier multi-threading tutorials we have learned how to solve producer consumer problem using [wait and notify](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html), and [BlockingQueue](http://javarevisited.blogspot.sg/2012/02/producer-consumer-design-pattern-with.html" \o "Click to open in a new window" \t "_blank) and in this tutorial we will learn *how to implement producer consumer design pattern using synchronous queue*. This class also supports an optional fairness policy for ordering waiting producer and consumer threads. By default, this ordering is not guaranteed. However, a queue constructed with fairness property set to true grants threads access in FIFO order.

## Producer Consumer using SynchronousQueue in Java

[](http://4.bp.blogspot.com/-flb9JJe-hkI/U5XHy7VwsCI/AAAAAAAABls/CXOO6feXhrM/s1600/Producer+consumer+solution+using+SynchronousQueue+in+Java.jpg)As I have said before, nothing is better than a producer consumer problem to understand [inter-thread communication](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html) in any programming language. In Producer consumer problem, one thread act as producer which produces event or task and other thread act as consumer. Shared buffer is used to transfer data from producer to consumer. Difficulty in solving producer consumer problem comes with edge cases e.g. producer must wait if buffer is full or consumer thread must wait if buffer is empty.  Later one was quite easy as blocking queue provides not only buffer to store data but also flow control to block thread callingput() method (PRODUCER) if buffer is full, and blocking thread calling take() method (CONSUMER) if buffer is empty.  In this tutorial, we will solve the same problem using SynchronousQueue, a special kind of [concurrent collection](http://javarevisited.blogspot.sg/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html) which has zero capacity.  
  
In following example, we have two threads which is named PRODUCER and CONSUMER (you should always name your threads, this is one of the best practice of writing concurrent application).  First thread, publishing cricket score, and second thread is consuming it. Cricket scores are nothing but a Stringobject here. If you run the program as it is you won't notice any thing different. In order to understand *how SynchronousQueue works*, and *how it solves producer consumer problem*, you either need to [debug this program in Eclipse](http://javarevisited.blogspot.sg/2011/07/java-debugging-tutorial-example-tips.html) or just start producer thread by commenting consumer.start(); If consumer thread is not running then producer will block at queue.put(event); call, and you won't see [PRODUCER] published event : FOUR. This happens because of special behaviour of SynchronousQueue, which guarantees that the thread inserting data will block until there is a thread to remove that data or vice-versa. You can test the other part of code by commenting producer.start(); and only starting consumer thread.  
import java.util.concurrent.SynchronousQueue;

/\*\*

\* Java Program to solve Producer Consumer problem using SynchronousQueue. A

\* call to put() will block until there is a corresponding thread to take() that

\* element.

\*

\* @author Javin Paul

\*/

public class SynchronousQueueDemo{

public static void main(String args[]) {

final SynchronousQueue<String> queue = new SynchronousQueue<String>();

Thread producer = new Thread("PRODUCER") {

public void run() {

String event = "FOUR";

try {

queue.put(event); // thread will block here

System.out.printf("[%s] published event : %s %n", Thread

.currentThread().getName(), event);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

};

producer.start(); // starting publisher thread

Thread consumer = new Thread("CONSUMER") {

public void run() {

try {

String event = queue.take(); // thread will block here

System.out.printf("[%s] consumed event : %s %n", Thread

.currentThread().getName(), event);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

};

consumer.start(); // starting consumer thread

}

}

Output:

[CONSUMER] consumed event : FOUR

[PRODUCER] published event : FOUR

If you have send the output carefully then you would have noticed that order of events are reversed. Seems [CONSUMER] thread is consuming data even before [PRODUCER] thread has produced it. This happens because by default SynchronousQueue doesn't guarantee any order, but it has a fairness policy, which if set to true allows access to threads in FIFO order. You can enable this fairness policy by passing true to [overloaded constructor](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) of SynchronousQueue i.e. new SynchronousQueue(boolean fair).

### Things to remember about SynchronousQueue in Java

Here are some of the important properties of this special blocking queue in Java. It's very useful to transfer data from one thread to another thread synchronously. It doesn't have any capacity and blocks until there is a thread on the other end.  
  
1) SynchronousQueue blocks until another thread is ready to take the element, one thread is trying to put.  
  
2) SynchronousQueue has zero capacity.  
  
3) SynchronousQueue is used to implement queuing strategy of  direct hand-off, where thread hands-off to waiting thread, else creates new one if allowed, else task rejected.  
  
4) This queue does not permit null elements, adding null elements will result in [NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).  
  
5) For purposes of other Collection methods (for example contains), a SynchronousQueue acts as an empty collection.  
  
6) You cannot peek at a synchronous queue because an element is only present when you try to remove it; Similarly you cannot insert an element (using any method) unless another thread is trying to remove it.  
  
7) You cannot iterate over SynchronousQueue as there is nothing to iterate.  
  
8) A SynchronousQueue constructed with fairness policy set to true grants threads access in FIFO order.  
  
That's all about **SynchronousQueue in Java**. We have seen some special property of this special concurrent collection, and learned how to solve classical producer consumer problem using SynchronousQueue in Java.  By the way calling it a Queue is bit confusing because it doesn't have any capacity to hold your element. Call to put() operation will not complete until there is a thread which is calling take() operation. It's better be a rendezvous point between threads to share objects. In other words, its a utility to synchronously share data between two threads in Java, probably a safer alternative of [wait and notify](http://java67.blogspot.sg/2013/03/difference-between-wait-vs-notify-vs-notifyAll-java-thread.html) methods.

**How to find all Pairs in Array of Integers whose Sum is equal to a given Number**Practising coding problems are very important to do well in any programming interview. You should at your best on data-structures like array, linked list, and string to clear any programming interview, and believe me you can not do this in one day or one week. It's rather long process of learning through coding, and that's where these small coding problems helps. Today, we are going to look at another interesting programming question from array; *write a program to find all pairs of integers whose sum is equal to a given number*. For example if input integer array is {2, 6, 3, 9, 11} and given sum is 9, output should be {6,3}. Sounds simple? may be, but this exactly question has appeared in technical interview at Amazon, Microsoft, Facebook and couple of other fortune five tech companies in past. Many of you might already heard about this question and some of you may already know the solution of this problem as well, but it's not enough to know just the answer. In a [programming interview](http://javarevisited.blogspot.sg/2011/06/top-programming-interview-questions.html), many things matter apart from correct solution. For example, first thing Interviewer look is whether candidate can ask right questions or not. So before jumping straight to coding, spare a second or two to think about problem and clear any doubt you may have. For example, you can ask following questions based upon problem statement given above :

* Does array contains only positive or negative numbers?
* What if same pair repeats twice, should we print it every time?
* Is reverse of pair is acceptable e.g. can we print both (4,1) and (1,4) if given sum is 5.
* Do we need to print only distinct pair? does (3, 3) is a valid pair for given sum of 6?
* How big the array is?

Many programmers afraid to ask questions instead they like to assume about it, but during coding interview IMHO it's always better to ask questions. First it shows that you have not mugged the answer and second it demonstrate that you have ability to think through a problem, which is a very important quality of any professional programmer. Now let's go back to question, for simplicity we can assume that we just need to print a pair of integers once or twice depending upon their occurrence, but pair has to be distinct, (2,2) or (3, 3) is not valid pair.

### 3 Solution to Find Pair Of Integers in Array whose Sum is Given Number

The first solution which comes in my mind is our friend brute-force, naive but genuine. You take one number from array and then [loop through array](http://java67.blogspot.sg/2013/08/how-to-iterate-over-array-in-java-15.html) and output pairs which is equal to given sum. You do this for all numbers in first array, as shown in following Java program :  
import java.util.Arrays;

/\*\*

\* Java Program to find pairs on integer array whose sum is equal to k

\* @author WINDOWS 8

\*/

public class ProblemInArray{

public static void main(String args[]) {

int[] numbers = { 2, 4, 3, 5, 7, 8, 9 };

int[] numbersWithDuplicates = { 2, 4, 3, 5, 6, -2, 4, 7, 8, 9 };

prettyPrint(numbers, 7);

prettyPrint(numbersWithDuplicates, 7);

}

/\*\*

\* Prints all pair of integer values from given array whose sum is is equal to given number.

\* complexity of this solution is O(n^2)

\*/

public static void printPairs(int[] array, int sum) {

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

int first = array[i];

for (int j = i + 1; j < array.length; j++) {

int second = array[j];

if ((first + second) == sum) {

System.out.printf("(%d, %d) %n", first, second);

}

}

}

}

/\*\*

\* Utility method to print input and output for better explanation.

\*/

public static void prettyPrint(int[] givenArray, int givenSum){

System.out.println("Given array : " + Arrays.toString(givenArray));

System.out.println("Given sum : " + givenSum);

System.out.println("Integer numbers, whose sum is equal to value : " + givenSum);

printPairs(givenArray, givenSum);

}

}

Output:

Given sum : 7

Integer numbers, whose sum is equal to value : 7

(2, 5)

(4, 3)

Given array : [2, 4, 3, 5, 6, -2, 4, 7, 8, 9]

Given sum : 7

Integer numbers, whose sum is equal to value : 7

(2, 5)

(4, 3)

(3, 4)

(-2, 9)

This solution is correct but it's time complexity is very hight, O(n^2), which means Interviewer will surely ask you to improve your answer and come up with solution whose complexity is either O(1), O(n) or O(nLog(n)). So let's dig deeper to improve this answer. In order to find two numbers in an array whose sum equals a given value, we probably don't need compare each number with other. What we can do here is to store all numbers in a [hashtable](http://javarevisited.blogspot.sg/2012/01/java-hashtable-example-tutorial-code.html" \o "Click to open in a new window" \t "_blank) and just check if it contains second value in a pair. For example, if given sum is 4 and one number in pair is 3, then other must be 1 or -7. Do you remember the first question we asked, if array only contains positive numbers then we don't need to check for negative values in Map. How is this solution better than previous one? It would require less comparisons. Only N to iterate through array and insert values in a Set because add() and contains() both O(1) operation in hash table. So total complexity of solution would be O(N). Here is a Java program which find the pair of values in the array whose sum is equal to k using Hashtable or Set. In this program we have also written a utility method to [generate random numbers in a given range in Java](http://javarevisited.blogspot.sg/2013/05/how-to-generate-random-numbers-in-java-between-range.html). You can use this method for testing with random inputs. By the way, random numbers are only good for demonstration, don't use them in your unit test. One more good thing you can learn from printPairsUsingSet() method is pre validation, checking if inputs are valid to proceed further.  
  
import java.util.Arrays;

import java.util.HashSet;

import java.util.Set;

/\*\*

\* Java Program to find two elements in an array that sum to k.

\*

\* @author WINDOWS 8

\*/

public class ArraySumUsingSet {

public static void main(String args[]) {

prettyPrint(getRandomArray(9), 11);

prettyPrint(getRandomArray(10), 12);

}

/\*\*

\* Given an array of integers finds two elements in the array whose sum is equal to n.

\* @param numbers

\* @param n

\*/

public static void printPairsUsingSet(int[] numbers, int n){

if(numbers.length < 2){

return;

}

Set set = new HashSet(numbers.length);

for(int value : numbers){

int target = n - value;

// if target number is not in set then add

if(!set.contains(target)){

set.add(value);

}else {

System.out.printf("(%d, %d) %n", value, target);

}

}

}

/\*

\* Utility method to find two elements in an array that sum to k.

\*/

public static void prettyPrint(int[] random, int k){

System.out.println("Random Integer array : " + Arrays.toString(random));

System.out.println("Sum : " + k);

System.out.println("pair of numbers from an array whose sum equals " + k);

printPairsUsingSet(random, k);

}

/\*\*

\* Utility method to return random array of Integers in a range of 0 to 15

\*/

public static int[] getRandomArray(int length){

int[] randoms = new int[length];

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

randoms[i] = (int) (Math.random()\*15);

}

return randoms;

}

}

Output:

Random Integer array : [0, 14, 0, 4, 7, 8, 3, 5, 7]

Sum : 11

pair of numbers from an array whose sum equals 11

(7, 4)

(3, 8)

(7, 4)

Random Integer array : [10, 9, 5, 9, 0, 10, 2, 10, 1, 9]

Sum : 12

pair of numbers from an array whose sum equals 12

(2, 10)

One more thing, here we are using HashSet but since [HashSet in Java internally uses HashMap](http://java67.blogspot.sg/2014/01/how-hashset-is-implemented-or-works-internally-java.html" \o "Click to open in a new window" \t "_blank), it would not make any difference if use either of those data structure.By the this solution has few constraints, first it would need additional space of order O(n) to store numbers in Hashtable or Set, so you need additional space which could be problem if array is very large (remember the question we asked before writing solution). For a large array, you need a solution which doesn't require additional space, also known as *in-place solution*. If interviewer will ask you how do you find if two values in an array sum to a given value without any additional space, first solution will also not work because it's complexity is too high and it would too long to [sort a large array](http://javarevisited.blogspot.sg/2012/01/sort-array-in-java-ascending-and.html). Asolution with complexity e.g. O(n), O(logN) or O(NLongN) should work though. A more efficient in-place solution would be to sort the array and use two pointers to scan through array from both direction i.e. beginning and end. If sum of both the values are equal to given number then we output the pair and advance them. If the sum of two numbers is less than k then we increase the left pointer, else if the sum is greater than k we decrement the right pointer, until both pointers meet at some part of the array. The complexity of this solution would be O(NlogN) due to sorting. Remember to use a in-place sorting algorithm like [quicksort](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html) to sort the array as we don't have additional space. Thankfully, Arrays.sort() method uses a two pivot quicksortalgorithm to sort array of primitives.

import java.util.Arrays;

import java.util.HashSet;

import java.util.Set;

/\*\*

\* Java Program to find all pairs on integer array whose sum is equal to k

\*

\* @author WINDOWS 7

\*/

public class PrintArrayPairs {

public static void main(String args[]) {

prettyPrint( new int[]{ 12, 14, 17, 15, 19, 20, -11}, 9);

prettyPrint( new int[]{ 2, 4, 7, 5, 9, 10, -1}, 9);

}

/\*\*

\* Given a number finds two numbers from an array so that the sum is equal to that number k.

\* @param numbers

\* @param k

\*/

public static void printPairsUsingTwoPointers(int[] numbers, int k){

if(numbers.length < 2){

return;

}

Arrays.sort(numbers);

int left = 0; int right = numbers.length -1;

while(left < right){

int sum = numbers[left] + numbers[right];

if(sum == k){

System.out.printf("(%d, %d) %n", numbers[left], numbers[right]);

left = left + 1;

right = right -1;

}else if(sum < k){

left = left +1;

}else if (sum > k) {

right = right -1;

}

}

}

/\*

\* Utility method to print two elements in an array that sum to k.

\*/

public static void prettyPrint(int[] random, int k){

System.out.println("input int array : " + Arrays.toString(random));

System.out.println("All pairs in an array of integers whose sum is equal to a given value " + k);

printPairsUsingTwoPointers(random, k);

}

}

Output :

input int array : [12, 14, 17, 15, 19, 20, -11]

All pairs in an array of integers whose sum is equal to a given value 9

(-11, 20)

input int array : [2, 4, 7, 5, 9, 10, -1]

All pairs in an array of integers whose sum is equal to a given value 9

(-1, 10)

(2, 7)

(4, 5)

That' all on this array based interview question to *find all pairs in an array of integers whose sum is equal to a given integer*. We have seen three ways to solve this problem starting from simplest brute-force solution to acceptable O(N) with additional space and O(NLogN) in-place. If anyone like to do some more practice, I would suggest to write [JUnit test cases](http://javarevisited.blogspot.sg/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) for this problem, given set of constraints that only unique pair needs to be printed even if array contains duplicated and find bugs on these solution. Alternatively, you can also try to solve it's cousin question, given an array of integers check whether there are 3 numbers that sum up to 0 or given number. Remember more fun is in journey than reaching the destination :)  
  
**Difference between getPath(), getCanonicalPath() and getAbsolutePath() of File in Java**File API is very important one in Java, it gives access of File system to Java programs. Though Java's file API is rich, there are lot of subtleties to know when you use them. One of the common query programmer's has about file path is difference between getPath(), getCanonicalPath() and getAbsolutePath() methods, why there are three methods to get file path and what happens if you call getPath() in place of getCanonicalPath(). By the way, before understanding difference between getPath(), getAbsolutePath() and getCanonicalPath() let's understand the concept behind this methods, i.e. *difference between path, absolute path, and canonical path*. In general, a path is way to get to a particular file or directory in a file system, it can be absolute (also known as full path) or relative e.g. relative to current location. Absolute path defines path from root of the file system e.g. C:\\ or D:\\ in Windows and from / in UNIX based operating systems e.g. [Linux](http://javarevisited.blogspot.sg/2011/06/10-examples-of-grep-command-in-unix-and.html) or [Solaris](http://javarevisited.blogspot.sg/2013/06/find-cpu-and-memory-used-by-java-solaris-prstat-command-example.html). Canonical path is little bit tricky, because *all canonical path is absolute, but vice-versa is not true*. It actually defines a unique absolute path to the file from root of the file system. For example, C://temp/names.txt is a canonical path to names.txt in Windows, and /home/javinpaul/test/names.txt is canonical path in Linux. On the other hand, there can be many absolute path to the same file, including the canonical path which has just seen. For example another absolute path to the same file  in Windows can be C://temp/./names.txt; similarly in UNIX /home/javinpaul/test/./names.txt is another absolute path to the same file. So you can say that, absolute path may contain [meta characters](http://javarevisited.blogspot.sg/2012/09/how-to-replace-escape-xml-special-characters-java-string.html) like . and .. to represent current and parent directory. In rest of this article, we will learn difference between getPath(), getAbsolutePath() andgetCanonical() Path by looking at values it return for a particular file.

## What is Absolute, Relative and Canonical Path

You often heard the term, absolute, canonical and relative path while dealing with files in UNIX, Windows, Linux or any file system. These are three common ways to reference any particular file in a script or program. If you are a programmer, writing script then you know how using absolute path can make your script rigid and in-flexible, infact using absolute path, infamously known as hard-coding path in script is one of the bad coding practice in programmer's dictionary. An absolute path is complete path to a particular file such as C:\temp\abc.txt. The definition of absolute pathname is also system dependent. On UNIX systems, a pathname is absolute if its prefix is "/". On Win32 systems, a pathname is absolute if its prefix is a drive specifier followed by "\\", or if its prefix is "\\".  
  
For example, we have two directories: temp and temp1 and test.txt file is in temp directory.  
C:\temp  
C:\temp1  
  
In Java under Windows, you may have the following possible absolute paths that refer to the same file test.txt.  
  
 C:\temp\test.txt

C:\temp\test.txt

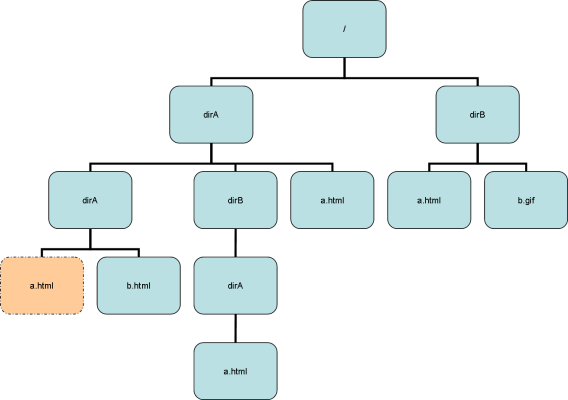
C:\temp\TEST.TXT

C:\temp\.\test.txt

C:\temp1\..\temp\test.txt

On the other hand, relative path is relative to the directory you are in, known as current directory. So if you are in the above directory, then if you reference file test.txt as relative, it assumes the same directory you are in. When you do ../ then it goes back one directory, also known as parent directory. Canonical paths are a bit harder. For starters, all canonical paths are absolute (but not all absolute paths are canonical). A single file existing on a system can have many different paths that refer to it, but only one canonical path. Canonical gives a unique absolute path for a given file. The details of how this is achieved are probably system-dependent. For the above example, we have one and only one canonical path: C:\temp\test.txt, Remember in Java you can UNIX style forward slash (/) use path separator or you can even get operating systems path separator using file.separator system property, a key to write truly [platform independent Java application](http://java67.blogspot.sg/2012/08/how-java-achieves-platform-independence.html).

### Difference between getPath(), getAbsolutePath() and getCanonicalPath() in Java

[](http://2.bp.blogspot.com/-M7smR5XqZtY/U-Tbvyb63XI/AAAAAAAABw0/0vZT6bxm4jE/s1600/Absolute+vs+Relative+Path+Java.png)

Once you understand *difference between absolute, canonical and relative path*, it would be very easy to differentiate between these three method, because they actually return path, absolute and canonical path. In short, here is key difference between them :

1. The first method, getPath()  return a String which denotes the path that is used to create associated File object, and it may be relative to current directory.
2. The second method, getAbsolutePath() returns the path string after resolving it against the current directory if it's relative, resulting in a fully qualified path.
3. The third method, getCanonicalPath() returns the path string after resolving any relative path against current directory, and removes any relative path element e.g. (. and ..), and any file system links to return a path which the file system considers the canonical means to reference the file system object to which it points.

Also remember that , each of above two method has a File equivalent which returns the corresponding File object e.g. getAbsoluteFile() and getCanonicalFile() which returns same thing.  
  
getPath() vs getAbsolutePath() vs getCanonicalPath()

The following example shows how there can be many different paths (and absolute paths) to the same file, which all have the exact same canonical path. Thus canonical path is useful if you want to know if two different paths point to the same file or not.  
import java.io.File;

/\*\*

\* Java program to show difference between path, absolute path and canonical

\* path related to files in Java. File API provides three methods to

\* java.io.File class getPath(), getAbsolutePath() and getCanonicalPath() and

\* this program just explain what those method returns.

\*

\* @author Javin Paul

\*/

public class PathDemo {

public static void main(String args[]) {

System.out.println("Path of the given file :");

File child = new File(".././Java.txt");

displayPath(child);

File parent = child.getParentFile();

System.out.println("Path of the parent file :");

displayPath(parent);

}

public static void displayPath(File testFile) {

System.out.println("path : " + testFile.getPath());

System.out.println("absolute path : " + testFile.getAbsolutePath());

try {

System.out.println("canonical path : " + testFile.getCanonicalPath());

} catch (Exception e) {

e.printStackTrace();

}

}

}

Output:

Path of the given file :

path : ..\.\Java.txt

absolute path : C:\Users\WINDOWS 8\workspace\Demo\..\.\Java.txt

canonical path : C:\Users\WINDOWS 8\workspace\Java.txt

Path of the parent file :

path : ..\.

absolute path : C:\Users\WINDOWS 8\workspace\Demo\..\.

canonical path : C:\Users\WINDOWS 8\workspace

## That's all about difference between getPath(), getAbsolutePath() and getCanonicalPath() in Java. In the course, we have also learned difference between path, absolute path and canonical path. What you need to remember is that, getPath() gives you the path on which File object is created, which may or may not be relative; getAbsolutePath() gives an absolute path to the file; and getCanonicalPath() gives you the unique absolute path to the file. It's worth noting that there can be a huge number of absolute paths that point to the same file, but only one canonical path. **What is GET and POST method in HTTP and HTTPS Protocol** **GET and POST method in HTTP and HTTPS Protocol**

**GET and POST method in HTTP and HTTPS** are two most popular methods used to transfer data from client to server using  HTTP(Hyper Text Transfer Protocol)  protocol. Both GET and POST can be used to send request and receive response but there are significant difference between them. **Difference between GET and POST in HTTP or HTTPS** is also a [popular interview question in JSP](http://javarevisited.blogspot.com/2011/10/jsp-interview-questions-answers-for.html) and any web programminginterview. Since HTML is independent of any web server technology like Java, ASP or PHP and HTTP is core protocol in space of internet, importance of clear understanding of GET and POST method can not be ignored. In this tutorial we will **What is GET HTTP Request**, What is POST HTTP Request, When to use GET and POST HTTP method and finally some *difference between GET and POST method in HTTP protocol*.

## **What is GET HTTP Request in http**

HTTP protocol supports several request method you can use while sending request using HTTP or HTTPS protocol. GET is one of them. As the name suggest GET method is to retrieve a page from HTTP Server. You can identify a GET request by looking method attribute on HTTP Request part. If you are using [Netbeans IDE](http://javarevisited.blogspot.com/2011/08/how-to-view-javadoc-in-netbeans-ide-70.html" \o "Click to open in a new window" \t "_blank) for Java web development you can enable HTTP Server monitor which can capture HTTP request and show details of request parameters, headers and other useful information. for GET HTTP request method will be GET for example almost all the URL which is accessible using link are accessed using HTTP Request. One important property of GET request is that any request parameter or query parameter is passed as [URL encoded string](http://javarevisited.blogspot.com/2012/01/url-rewriting-url-encoding-in-servlet.html), appended using "?" character which makes it non secure because whatever information you pass in URL String is visible to everybody. Though GET method has some very interesting and powerful use cases which we will seen in next section : **When to use GET HTTP Request**?

## **When to use HTTP GET request**

As I said **GET method** is not secure and hence not a suitable choice for transferring confidential data but GET method is extremely useful for retrieving static content from web server. here are some examples where a using GET method make sense:

1) There is no side effect of repeated request. for example clicking a link which points to another page. it doesn't matter if you click the link twice or thrice , This also gives chance browser of server to catch the response for faster retrieval.

2) You are not passing any sensitive and confidential information. instead you just passing some configuration data or session id.

3) You want URL pointed by HTTP GET request to be bookmark-able.

4) Data requires to be sent to Server is not large and can safely accommodated in maximum length of URL supported by all browser. In general different browser has different character limit for URL length but having it under limit is good choice.

## **What is POST HTTP method**

**POST HTTP request** is denoted by method: POST in HTTP request. In POST method data is not sent as part of URL string to server instead in POST, *data is sent as part of message body*. Almost all [authentication request](http://javarevisited.blogspot.com/2011/11/ldap-authentication-active-directory.html) is sent via POST method in HTTP world. POST method is secure because data is not visible in URL String and can be safely encrypted using HTTPS for further security. All sensitive and confidential information sent to be server must go on POST request and via HTTPS (HTTP with SSL). POST method is also used for submitting information to server, any information which can alter state of application like adding item into shopping cart, making payments etc. here are some examples where you should consider using POST method in HTTP request:

1) Use POST if you are sending large data which can not be fit into URL in case of GET.

2) Use POST method if you are passing sensitive and confidential information to server e.g. user\_id, password, account number etc.

3) Use POST method if you are submitting data which can alter state of application e.g. adding items into cart for passing that cart for payment processing.

4) Use POST if you are writing secure application and don't want to show query parameters in URL.

## **Difference between GET and POST method in HTTP Protocol**

Most of the *difference between GET and POST* has been already discussed in there respective section. It all depends upon requirement when you want to choose GET and POST and knowledge of these differences help you to make that decision.

1) GET method passes request parameter in URL String while POST method passes request parameter in request body.

2) GET request can only pass limited amount of data while POST method can pass large amount of data to server.

3) GET request can be bookmarked and cached unlike POST requests.

4) GET is mostly used for view purpose (e.g. SQL SELECT) while POST is mainly use for update purpose (e.g. SQL INSERT or UPDATE).

**How to Use Locks in Multi-threaded Java Program**Many Java programmers confused themselves like hell while writing multi-threaded Java programs e.g. where to synchronized? Which Lock to use? What Lock to use etc. I often receive request to explain about *how to use Locks in Java*, so I thought to write a simple Java program, which is multi-threaded and uses rather new Lock interface. Remember Lock is your tool to guard shared resource which can be anything e.g. database, File system, a Prime number Generator or a Message processor. Before using Locks in Java program, it’s also better to learn some basics. Lock is an interface from java.util.concurrent package. It was introduced in JDK 1.5 release as an alternative of synchronized keyword. If you have never written any multi-threading program, then I suggest first start with synchronized keyword because it’s easier to use them. Once you are familiar with working of multi-threading program e.g. How threads share data, [how inter thread communication works](http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html), you can start with Lock facility. As I told you Lock is an interface, so we cannot use it directly, instead we need to use its implementation class. Thankfully Java comes with two implementation of java.util.concurrent.locks.Lock interface, ReentrantLock and ReentrantReadWriteLock, later provides two more inner implementation known as ReentrantReadWriteLock.ReadLock and ReentrantReadWriteLock.WriteLock. For our simple multi-threaded Java program's purpose ReentrantLock is enough.  
Here is the idiom to use Locks in Java : 

Lock l = ...;

l.lock();

**try** {

// access the resource protected by this lock

} **finally** {

l.unlock();

}

You can see that Lock is used to protect a resource, so that only one thread can access it at a time. Why we do that? to make sure our application behave properly. For example we can use Lock to protect a counter, whose sole purpose is to return a count incremented by one, when anyone calls its getCount() method. If we don't protect them by [parallel access of thread](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html), then it’s possible that two thread receives same count, which is against the program's policies. Now, coming back to semantics, we have used lock() method to acquire lock and unlock() method to release lock. Always remember to release lock in finally block, because every object has only one lock and if a thread doesn't release it then no one can get it, which may result in your program hung or threads going into deadlock. That's why I said that synchronized keyword is simpler than lock, because Java itself make sure that lock acquired by thread by entering into synchronized block or method is released as soon as it came out of the block or method. This happens even if thread came out by throwing exception, this is also we have [unlock code in finally block](http://javarevisited.blogspot.com/2012/11/difference-between-final-finally-and-finalize-java.html), to make sure it run even if try block throws exception or not. In next section we will see example of our multi-threaded Java program, which uses Lock to protect shared Counter.

## Java Lock and ReentrantLock Example

Here is a sample Java program, which uses both Lock and ReentrantLock to protect a shared resource. In our case it’s an object, a counter's object. Invariant of Counter class is to return a count incremented by 1 each time someone calls getCount() method. Here for testing three threads will call getCount() method simultaneously but guard provided by Lock will prevent shared counter. As an exercise you can also implement same class using [synchronized keyword](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html). Here is complete code :

**import** **java.util.concurrent.locks.Lock**;

**import** **java.util.concurrent.locks.ReentrantLock**;

/\*\*

\*

\* Java Program to show how to use Locks in multi-threading

\* e.g. ReentrantLock, ReentrantReadWriteLock etc.

\*

\* @author Javin Paul

\*/

**public** **class** **LockDemo** {

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

// Let's create a counter and shared it between three threads

// Since Counter needs a lock to protect its getCount() method

// we are giving it a ReentrantLock.

**final** Counter myCounter = **new** Counter(**new** ReentrantLock());

// Task to be executed by each thread

Runnable r = **new** Runnable() {

**@Override**

**public** **void** **run**() {

System.out.printf("Count at thread %s is %d %n",

Thread.currentThread().getName(), myCounter.getCount());

}

};

// Creating three threads

Thread t1 = **new** Thread(r, "T1");

Thread t2 = **new** Thread(r, "T2");

Thread t3 = **new** Thread(r, "T3");

//starting all threads

t1.start();

t2.start();

t3.start();

}

}

**class** **Counter** {

**private** Lock lock; // Lock to protect our counter

**private** **int** count; // Integer to hold count

**public** **Counter**(Lock myLock) {

**this**.lock = myLock;

}

**public** **final** **int** **getCount**() {

lock.lock();

**try** {

count++;

**return** count;

} **finally** {

lock.unlock();

}

}

}

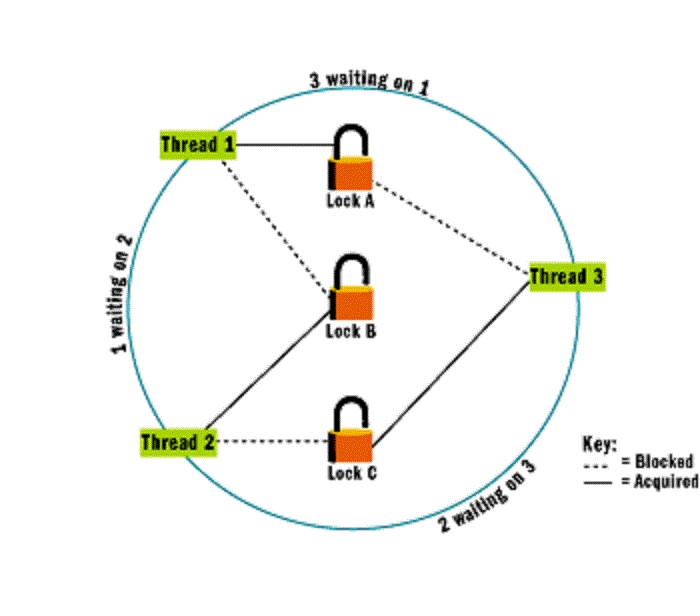
**Output:**

Count at thread T1 is **1**

Count at thread T2 is **2**

Count at thread T3 is **3**

You can even put a long loop inside Runnable's run() method to call getCount() numerous time, if you see a duplicate means there is a problem with your code, but without any duplicate means it’s working fine.

[](http://1.bp.blogspot.com/-7MKR8YVaC9c/VD0v4-AFOII/AAAAAAAACCQ/MkIqUKhlf2g/s1600/Lock+Example+in+Java.gif)

## Common Mistakes made by beginners while using Locks in Java

Here are some of the common mistakes I have observed by looking at Java beginners lock related code :  
  
1) Instead of sharing lock they provide different locks to each thread. This often happens to them unknowingly because they usually put the lock and guarded block inside Runnable, and they pass two [different instances of Runnable](http://java67.blogspot.com/2012/08/what-is-thread-and-runnable-in-java.html) to two different threads e.g. whereSimpleLock is a Runnable, as shown below :

Thread firstThread = **new** Thread(**new** SimpleLock());

Thread secondThread = **new** Thread(**new** SimpleLock());

**class** **SimpleLock** **implements** Runnable {

**private** Lock myLock = **new** ReentrantLock();

**public** **void** **printOutput**() {

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

}

**public** **void** **run**() {

**if** (myLock.tryLock()) {

myLock.lock();

printOutput();

}**else**

System.out.println("The lock is not accessible.");

}

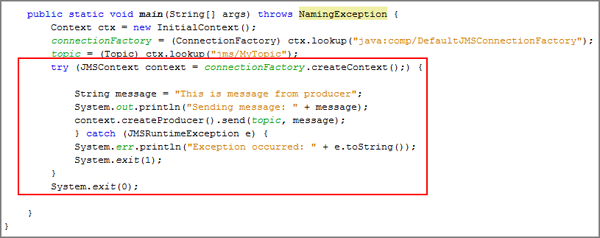
}

Since here myLock is instance variable, each instance of SimpleLock has their own myLock instance, which means firstThread and secondThread are using different lock and they can run protected code simultaneously.  
  
2) Second mistake Java beginners do is forget to call unlock() method, just like above example. without calling unlock() method, Thread will not release its lock and another thread waiting for that lock will never get that. Nothing will happen in this test program, but once you write this kind of code in real application, you will see nasty issues like deadlock, starvation and data corruption.  By the way Lock interface also offers several advantages over synchronized keyword, check [here](http://javarevisited.blogspot.com/2013/03/reentrantlock-example-in-java-synchronized-difference-vs-lock.html) to learn more.  
  
That's all about **how to use Locks in multi-threaded Java program for synchronization**. Let me know if you have any difficult understanding Locks in Java or anything related to multi-threading, Will be glad to help you. For further reading, you can explore Java documentation of Lock interface and it's various implementation classes  
  
[**Don't use System.exit() on Java Web Application**](http://javarevisited.blogspot.com/2014/11/dont-use-systemexit-on-java-web-application.html)

I have recently come across a code snippet, where programmer was using System.exit() if application failed to acquire necessary resource after couple of retry. His reasoning was that since, application cannot function, if essential resources like database is not available or there is no disk space to write records in File system. Ok, I hear you; but System.exit() in Java Web application, which runs inside either web server or application server, which itself is Java program is not a good idea at all. Why? because invoking System.exit() kills your JVM, invoking this from Tomcat or Jetty, will not only kill your application but most likely server itself. This can be potentially dangerous, if that server also host other critical application, which is not uncommon at all. As per my experience, System.exit() calls are quite common in overly broad try-catch blocks in web application start-up code that loads [environment variables](http://javarevisited.blogspot.com/2012/08/how-to-get-environment-variables-in.html), [properties files](http://javarevisited.blogspot.com/2012/12/how-to-create-and-modify-properties.html), connect to MQ Series, establishes database connection, opens socket connections, etc. This is still ok, if you are writing core Java based server, where each application has their own JVM, but with web application deployed on Tomcat, JBoss, WebSphere, Weblogic or any other application server, using System.exit() is big mistake. In worst case can result in outage for lots of other critical application. On the other hand, there are ways to prevent your web application from someone else’s mistake, by enabling Security Manager. System.exit() and Runtime.exit() both goes through the security manager. Enabling Security manager will catch these calls and reduce them into an exception rather than shutting down the whole VM. It's not difficult to enable the security manager in most application servers, Tomcat, JBoss both has documented steps to enable security Manager.

## Why you should not use System.exit() in Java Web application

I think it's well known for senior Java developer that System.exit() shouldn't be used except in e.g. command-line tools; Many beginner Java programmers, though familiar with System.exit(), may not know that using them in a Servlet/JSP code can result in shutdown of server itself, so if you have of any of them in your team them take some time to educate them about coding in Java web application. In fact coding a Java based web application is totally different then coding a core Java application e.g. core concerns like threading, object pooling, parsing are all done by Web server and it's prohibited for application code to create threads. In fact use of [ThreadLocal variable](http://javarevisited.blogspot.com/2012/05/how-to-use-threadlocal-in-java-benefits.html) can create memory leak in Java web-app, so coding in web-app require more caution than in core Java application. By the way there are other reasons, *why using System.exit is absolutely terrible*. Especially when you dealing with un-managed resources, if you don't release resources properly and hope that OS will do the clean-up for you, then it could lead to a temporary resource leak, until OS really clean stuff created by your Java application.

[](http://3.bp.blogspot.com/-m7iisqg5y4A/VHNAYgHMpZI/AAAAAAAACHU/bD03RFTVI6g/s1600/Using+System.exit+in+Java+Web+Application.png)

What does this all mean? Do System.exit() has any legitimate use? Of course there are many cases where use of System.exit is imperative. Sometime you really want to close your [JVM](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html) once done, especially if it spawned from scheduling software like Control-M or Autosys. For example command line Java applications and scheduled tasks, can use System.exit(). Ideally, if you are sure that calling System.exit() will not have any side effect, you can use it. Here is a sample use of system.exit() in core Java application :  
**public** **static** **void** **main**( String[] args ){

**try**

{

processFxRatesFileAndPutIntoDatabase();

} **catch** ( Exception e )

{

logError(e);

// if you need to exit with a failed status then System.exit(statusCode) is fine here.

// otherwise program will complete successfully, once it return from main() method

}

}

Remember, if you are using Java program as command line tool then you can return System.exit(0) or System.exit(1) as whether it succeeded or failed. A non-zero exit status code, usually indicates abnormal termination. You can return different error codes to indicate different errors e.g. bad arguments, can't find file, could not connect to database etc. I hope you notice difference between using System.exit(1) and letting the Java program complete successfully. When java platform will terminate with System.exit() it will do so with a non-zero status (as long as the main thread ends and there are no running [daemon threads](http://javarevisited.blogspot.com/2012/03/what-is-daemon-thread-in-java-and.html)).  
  
  
That’s all about **Why you should not use System.exit() inside Java application.** It can be dangerous and potentially shutdown whole server, which is hosting other critical Java application. For example if you have more than one application in tomcat and one of them call System.exit() then whole tomcat will be shutdown and all other web application will also be shutdown. As a safety net or Insurance, you must enable Security Manager of your Server. It catches a lot of security problems! Including intentional and unintentional call to System.exit() or Runtime.exit(), turning it into an exception rather than shutting down the whole JVM. You are relatively safe to use System.exit() in command line Java application, as they run on their own JVM and can be used to signal a success and failure by returning different status code in case of failure.