1. Compile time runtime exceptions and list of exceptions
2. Types of class loader

Java class loaders can be broadly classified into below categories:

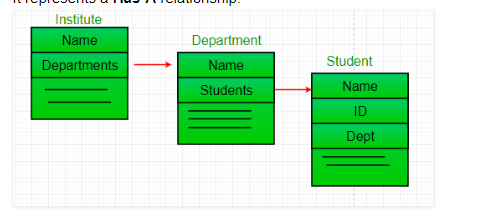
* Bootstrap Class Loader  
  Bootstrap class loader loads java’s core classes like java.lang, java.util etc. These are classes that are part of java runtime environment. Bootstrap class loader is native implementation and so they may differ across different JVMs.
* Extensions Class Loader  
  JAVA\_HOME/jre/lib/ext contains jar packages that are extensions of standard core java classes. Extensions class loader loads classes from this ext folder. Using the system environment propery java.ext.dirs you can add ‘ext’ folders and jar files to be loaded using extensions class loader.
* System Class Loader  
  Java classes that are available in the java classpath are loaded using System class loader.

1. **Difference between Composition, Aggregation and Association in OOP?** ([answer](http://javarevisited.blogspot.sg/2014/02/ifference-between-association-vs-composition-vs-aggregation.html))  
   If two objects are related to each other, they are said to be associated with each other. Composition and Aggregation are two forms of association in object-oriented programming. The composition is stronger association than Aggregation. In Composition, one object is OWNER of another object while in Aggregation one object is just USER of another object. If an object A is composed of object B then B doesn't exist if A ceased to exists, but if object A is just an aggregation of object B then B can exists even if A ceased to exist.

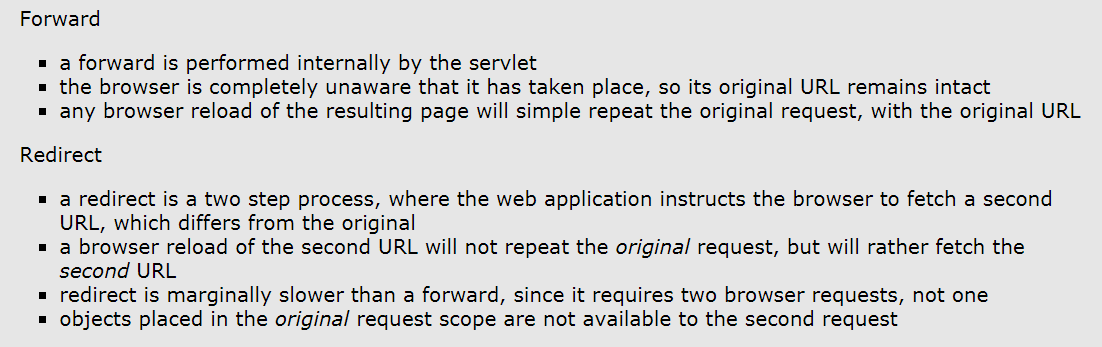
<https://www.geeksforgeeks.org/association-composition-aggregation-java/>

Is a Relation : A Library has books (Without books there is no library close relation)

Has a Relation



1. Rest –Representation state transfer
2. Suitable Collection for shopping cart
3. Hash map with same hash code with multiple objects
4. How you handled out of space memory issue
5. Action Forward vs Action Redirect

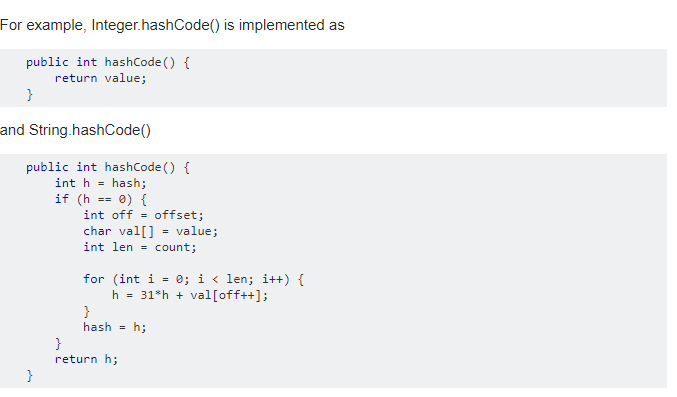


1. How the hash value calculated

// bitwise xor

// 0101 ^ 0111=0010

Object.hashcode(key) ^ Object.hashcode(value);



1. Complexity of collections
2. Diff preparedstatement,callable statement and statement in jdbc
3. Array.length vs Arraylist.size()

use length attribute to get number of elements in a array, also known as length, and for same thing in Collection classes e.g. [ArrayList](http://java67.blogspot.sg/2012/11/java-arraylist-example-contains-add-set.html), [Vector](http://java67.blogspot.sg/2012/09/arraylist-vs-vector-in-java-interview.html), use size() method

1. Multiple start() method in threads

It will throw illegal state modification exception

1. Difference between an array and ArrayList in Java

Array length is fixed and array list support primitive datatype.

1. Semaphore

It has accuire and release methods , it will allow only fixed count of connections only in threads .

It uses for database connections

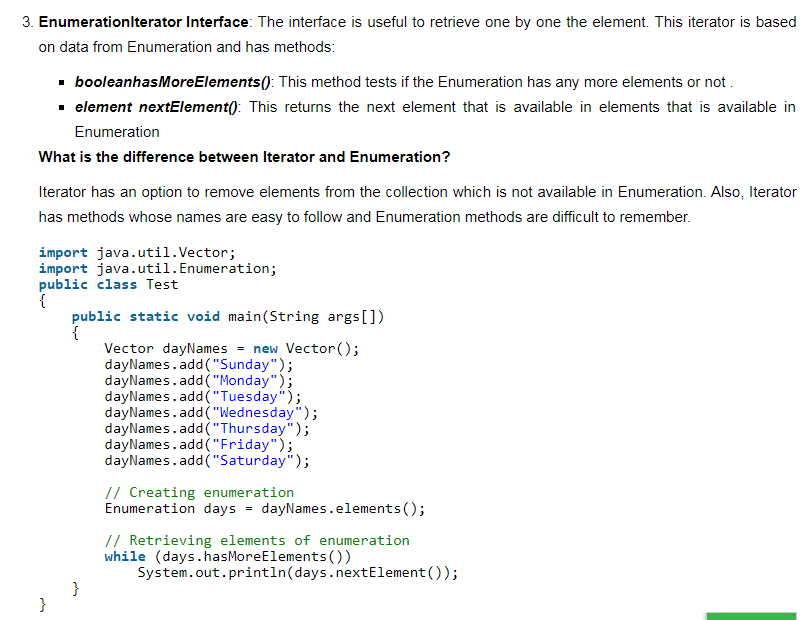
1. Count down latch vs cyclic barrier

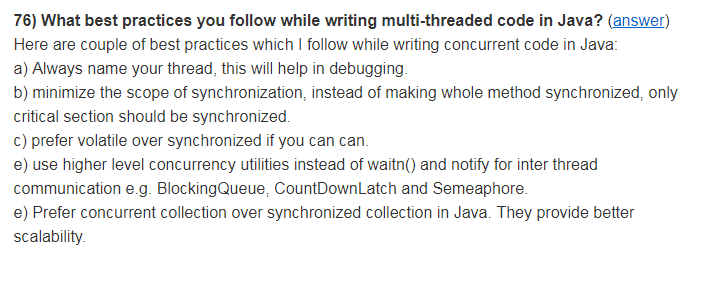
A CountDownLatch can be used only once in a program(until it’s count reaches 0).

A CyclicBarrier can be used again and again once all the threads in a barriers is released.

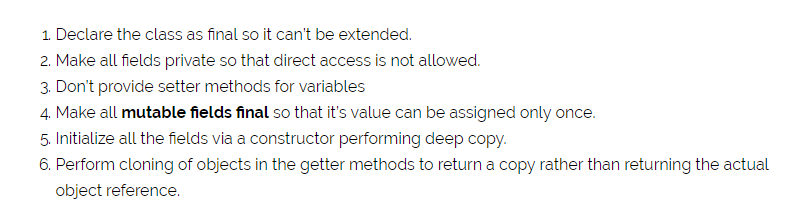
1. Methods in collection util (reverse,syncronise,umodifiedlist)
2. Diff between collection and collections

Collection is interface and collections is a util class

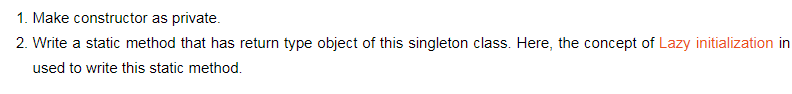
1. Object level locking and class level locking
2. How do you find memory usage from Java program? How much percent of the heap is used?  
   You can use memory related methods from java.lang.Runtime class to get the free memory, total memory and maximum heap memory in Java.  By using these methods, you can find out how many percents of the heap is used and how much heap space is remaining. Runtime.freeMemory() return amount of free memory in bytes, Runtime.totalMemory() returns total memory in bytes and Runtime.maxMemory() returns maximum memory in bytes.  
   
3. Spring bean life cycle



1. Transaction management
2. Things need to be done for basic operation with database
3. Design patterns used in Spring (Singleton,FatoryBean,Abstract Bean)
4. Concurrency package
5. How to make object as immutable



1. How to make object as singleton



1. How to make immutable objects as read only
2. How thread creations works in JVM
3. Joins in SQL
4. When to use Thread and Runnable class
5. Can we give private modifier in subclass to public modifier in super class

No

1. How singleton works in multi thread env
2. String reversal in linked list



1. Nth highest salary in sql for all dB

select \* from Products p1 where n-1 = (select count(Distinct(price)) from Products p2 where p2.price>p1.price)

1. Swagger documentation customization
2. Internal implementation of Concurrency hashmap
3. When to use concurrency hasmap vs normal hashmap
4. Functional interface usage ?
5. SOLID design principles

# Design Principles [S.O.L.I.D.] in Java

June 7, 2013 by Lokesh Gupta

Classes are the building blocks of your java application. If these blocks are not strong, your building (i.e. application) is going to face the tough time in future. This essentially means that not so well-written can lead to very difficult situations when the application scope goes up or application faces certain design issues either in production or maintenance.

On the other hand, set of well designed and written classes can speed up the coding process by leaps and bounds, while reducing the number of bugs in comparison.

In this post, I will list down 5 most recommended design principles, you should keep in mind, while writing your classes. These design principles are called SOLID, in short. They also form the [**best practices**](https://howtodoinjava.com/category/best-practices/) to be followed for designing your application classes.

**Table Of Contents**

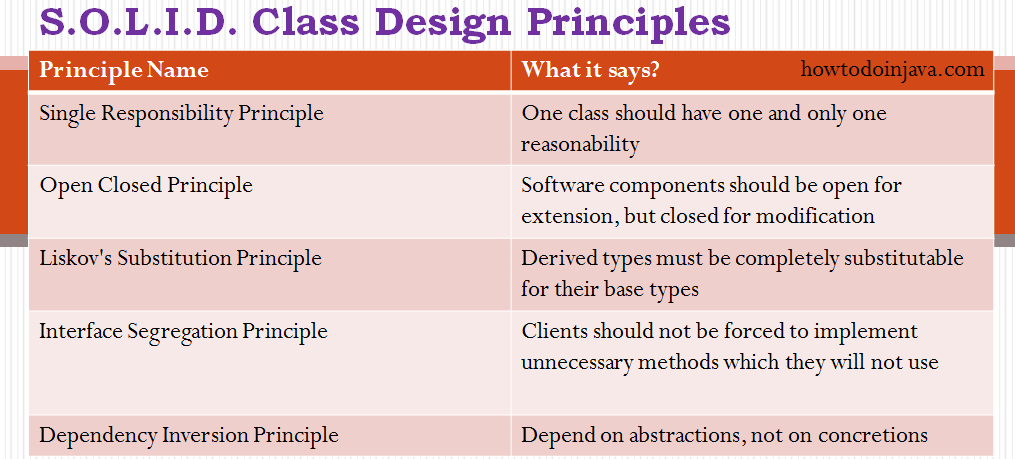
[Single Responsibility Principle](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#SRP)

[Open Closed Principle](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#OCP)

[Liskov's Substitution Principle](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#LSP)

[Interface Segregation Principle](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#ISP)

[Dependency Inversion Principle](https://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#DI)

5 java class design principles

Lets drill down all of them one by one.

## ****Single Responsibility Principle****

The name of the principle says it all:

**"One class should have one and only one responsibility"**

In other words, you should write, change and maintain a class for only one purpose. If it is model class then it should strictly represent only one actor/ entity. This will give you the flexibility to make changes in future without worrying the impacts of changes for another entity.

Similarly, If you are writing service/manager class then it should contain only that part of method calls and nothing else. Not even utility global functions related to module. Better separate them in another globally accessible class file. This will help in maintaining the class for that particular purpose, and you can decide the visibility of class to specific module only.

## ****Open Closed Principle****

This is second important rule which you should keep in mind while designing your application. It says:

**"Software components should be open for extension, but closed for modification"**

What does it mean?? It means that your classes should be designed such a way that whenever fellow developers wants to change the flow of control in specific conditions in application, all they need to extend your class and override some functions and that’s it.

If other developers are not able to design desired behavior due to constraints put by your class, then you should reconsider changing your class. I do not mean here that anybody can change the whole logic of your class, but he/she should be able to override the options provided by software in unharmful way permitted by software.

For example, if you take a look into any good framework like struts or spring, you will see that you can not change their core logic and request processing, BUT you modify the desired application flow just by extending some classes and plugin them in configuration files.

## ****Liskov’s Substitution Principle****

This principle is a variation of previously discussed open closed principle. It says:

**"Derived types must be completely substitutable for their base types"**

It means that the classes fellow developer created by extending your class should be able to fit in application without failure. I.e. if a fellow developer poorly extended some part of your class and injected into framework/ application then it should not break the application or should not throw fatal [**exceptions**](https://howtodoinjava.com/best-practices/java-exception-handling-best-practices/).

This can be insured by using strictly following first rule. If your base class is doing one thing strictly, the fellow developer will override only one feature incorrectly in worst case. This can cause some errors in one area, but whole application will not do down.

## ****Interface Segregation Principle****

This principle is my favorite one. It is applicable to interfaces as single responsibility principle holds to classes. It says:

**"Clients should not be forced to implement unnecessary methods which they will not use"**

Take an example. Developer Alex created an interface Reportable and added two methods generateExcel() and generatedPdf(). Now client ‘A’ wants to use this interface but he intend to use reports only in PDF format and not in excel. Will he achieve the functionality easily.

NO. He will have to implement two methods, out of which one is extra burden put on him by designer of software. Either he will implement another method or leave it blank. So are not desired cases, right??

So what is the solution? Solution is to create two interfaces by breaking the existing one. They should be like PdfReportable and ExcelReportable. This will give the flexibility to user to use only required functionality only.

## ****Dependency Inversion Principle****

Most of us are already familiar with the words used in principle’s name. It says:

**"Depend on abstractions, not on concretions"**

In other words. you should design your software in such a way that various modules can be separated from each other using an abstract layer to bind them together. The classical use of this principle of **[BeanFactory](https://howtodoinjava.com/spring/spring-core/different-spring-3-ioc-containers-with-example/" \o "Different spring 3 IoC containers with example" \t "_blank)** in [**spring framework**](https://howtodoinjava.com/java-spring-framework-tutorials/). In spring framework, all modules are provided as separate components which can work together by simply injected dependencies in other module. They are so well closed in their boundaries that you can use them in other software modules apart from spring with same ease.

This has been achieved by dependency inversion and open closed principles. All modules expose only abstraction which is useful in extending the functionality or plugin in another module.

These were **five class design principles** which makes the best practices to be followed to design your application classes. Let me know of your thoughts.

1. Data encapsulations vs abstraction
2. Searalize the static methods

No you can serialize static methods

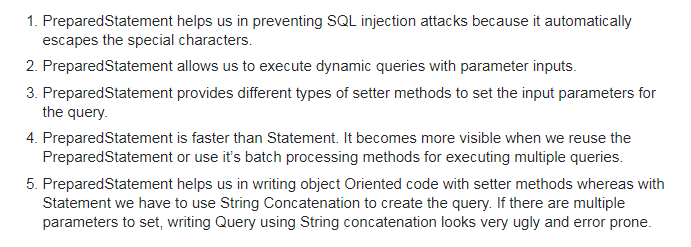
1. Jvm architecture
2. Fail fast vs fail safe
3. Demon thread
4. How you know object is available for garbage collection

Object reference changed or allocation object as null

1. Hasmap internal working
2. Return type of hash code

int

1. Comparable vs comparator
2. Big annotation of Hash map
3. Diff of java version and features
4. Thread pooling and disadvatages (memoryleak, deadlock)
5. Why the performance of cuncurrancy hashmap is fast
6. What is the difference between CyclicBarrier and CountDownLatch in Java
7. Methods of Object (clone,getclass,wait,equals,notify,notifyall,hascode,tostring,finalize)
8. Prepared statement vs statement (SQL injection)



1. What is differnance between sumbit() and execute() in threadpool?

Both methods are ways to submit a task to thread pools but there is a slight difference between them. execute(Runnable command) is defined in Executor interface and executes given task in future, but more importantly, it does not return anything. Its return type is void. On other hand submit() is an overloaded method, it can take either Runnable or Callable task and can return Future object which can hold the pending result of computation. This method is defined on ExecutorService interface, which extends Executor interface, and every other thread pool class e.g. ThreadPoolExecutor or ScheduledThreadPoolExecutor gets these methods.

1. Why is Java called the ‘Platform Independent Programming Language’?

Platform independence means that execution of your program does not dependent on type of operating system(it could be any : Linux, windows, Mac ..etc). So compile code only once and run it on any System (In C/C++, we need to compile the code for every machine on which we run it). Java is both compiler(javac) and interpreter(jvm) based lauguage. Your java source code is first compiled into byte code using javac compiler. This byte code can be easily converted to equivalent machine code using JVM. JVM(Java Virtual Machine) is available in all operating systems we install. Hence, byte code generated by javac is universal and can be converted to machine code on any operating system, this is the reason why java is platform independent.

1. What is livelock and deadlock?

This question is extension of previous interview question. A livelock is similar to a deadlock, except that the states of the threads or processes involved in the livelock constantly change with regard to one another, without any one progressing further. Livelock is a special case of resource starvation. A real-world example of livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress because they both repeatedly move the same way at the same time. In short, the main difference between livelock and deadlock is that in former state of process change but no progress is made.

**45)Explain Final keyword in java?**

Final keyword in java is used to restrict usage of variable, class and method.  
   
Variable: Value of Final variable is constant, you can not change it.  
Method: you can’t override a Final method.  
Class: you can’t inherit from Final class.

Refer [this](http://quiz.geeksforgeeks.org/final-keyword-java/) for details

**46)When is the super keyword used?**  
   
super keyword is used to refer:

* immediate parent class constructor,
* immediate parent class variable,
* immediate parent class method.

Refer [this](http://quiz.geeksforgeeks.org/super-keyword/) for details.

**47)What is the difference between StringBuffer and String?**

String is an Immutable class, i.e. you can not modify its content once created. While StringBuffer is a mutable class, means you can change its content later. Whenever we alter content of String object, it creates a new string and refer to that,it does not modify the existing one. This is the reason that the performance with StringBuffer is better than with String.  
Refer [this](https://www.geeksforgeeks.org/g-fact-27-string-vs-stringbuilder-vs-stringbuffer/) for details.

**48)Why multiple inheritance is not supported in java?**

Java supports multiple inheritance but not through classes, it supports only through its interfaces. The reason for not supporting multiple inheritance is to avoid the conflict and complexity arises due to it and keep Java a Simple Object Oriented Language. If we recall [this in C++](https://www.geeksforgeeks.org/multiple-inheritance-in-c/), there is a special case of multiple inheritance (diamond problem) where you have a multiple inheritance with two classes which have methods in conflicts. So, Java developers decided to avoid such conflicts and didn’t allow multiple inheritance through classes at all.

**49)Can a top level class be private or protected?**

Top level classes in java can’t be private or protected, but inner classes in java can. The reason for not making a top level class as private is very obvious, because nobody can see a private class and thus they can not use it. Declaring a class as protected also doesn’t make any sense. The only difference between default visibility and protected visibility is that we can use it in any package by inheriting it. Since in java there is no such concept of package inheritance, defining a class as protected is no different from default.

**50)What is the difference between ‘throw’ and ‘throws’ in Java Exception Handling?**

Following are the differences between two:

* throw keyword is used to throw Exception from any method or static block whereas throws is used to indicate that which Exception can possibly be thrown by this method
* If any method throws checked Exception, then caller can either handle this exception(using try catch block )or can re throw it by declaring another ‘throws’ clause in method declaration.
* throw clause can be used in any part of code where you feel a specific exception needs to be thrown to the calling method

E.g.  
**throw**  
throw new Exception(“You have some exception”)  
throw new IOException(“Connection failed!!”)  
**throws**  
throws IOException, NullPointerException, ArithmeticException

**51)What is finalize() method?**  
   
Unlike c++ , we don’t need to destroy objects explicitly in Java. ‘[Garbage Collector](https://www.geeksforgeeks.org/garbage-collection-java/)‘ does that automatically for us. Garbage Collector checks if no references to an object exist, that object is assumed to be no longer required, and the memory occupied by the object can be freed. Sometimes an object can hold non-java resources such as file handle or database connection, then you want to make sure these resources are also released before object is destroyed. To perform such operation Java provide protected void finalize() in object class. You can override this method in your class and do the required tasks. Right before an object is freed, the java run time calls the finalize() method on that object. Refer [this](https://www.geeksforgeeks.org/garbage-collection-java/) for more details.

**52)Difference in Set and List interface?**

Set and List both are child interface of Collection interface. There are following two main differences between them

* List can hold duplicate values but Set doesn’t allow this.
* In List interface data is present in the order you inserted but in the case of Set insertion order is not preserved.

**53)What will happen if you put System.exit(0) on try or catch block? Will finally block execute?**  
   
By Calling System.exit(0) in try or catch block, we can skip the finally block. System.exit(int) method can throw a SecurityException. If Sysytem.exit(0) exits the JVM without throwing that exception then finally block will not execute. But, if System.exit(0) does throw security exception then finally block will be executed.  
**Can we** [**Overload or Override static methods in java**](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/) **?**

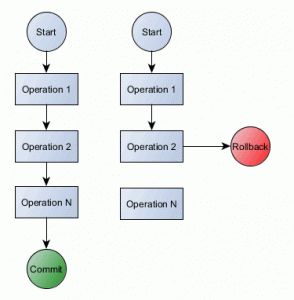
* **Overriding :** Overriding is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class) at runtime.
* **Overloading:** Overloading is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures, especially number of input parameters and type of input paramaters.
* **Can we overload static methods?**The answer is **‘Yes’**. We can have two ore more static methods with same name, but differences in input parameters
* **Can we Override static methods in java?**We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is **‘No’**. Static methods cannot be overridden because method overriding only occurs in the context of dynamic (i.e. runtime) lookup of methods. Static methods (by their name) are looked up statically (i.e. at compile-time).

Read [more](https://www.geeksforgeeks.org/can-we-overload-or-override-static-methods-in-java/)

1. ACID Prinicples

Transactions are omnipresent in today’s enterprise systems, providing data integrity even in highly concurrent environments. So let’s get started by first defining the term and the context where you might usually employ it.

A transaction is a collection of read/write operations succeeding only if all contained operations succeed.

[](https://vladmihalcea.files.wordpress.com/2014/01/transaction-workflow1.gif)

Inherently a transaction is characterized by four properties (commonly referred as ACID):

1. Atomicity
2. Consistency
3. Isolation
4. Durability

In a relational database, every SQL statement must execute in the scope of a transaction. Without defining the transaction boundaries explicitly, the database is going to use an implicit transaction which is wraps around every individual statement. The implicit transaction begins before the statement is executed and end (commit or rollback) after the statement is executed.  
The implicit transaction mode is commonly known as [autocommit](http://en.wikipedia.org/wiki/Autocommit).

For an enterprise application, the auto-commit mode is something you’d generally want to avoid since it has serious performance penalties, and it doesn’t allow you to include multiple [DML](http://en.wikipedia.org/wiki/Data_manipulation_language)operations in a single atomic Unit of Work.

It’s very important to understand those, hence we will discuss each and every one of them as follows.

# Atomicity

Atomicity takes individual operations and turns them into an all-or-nothing unit of work, succeeding if and only if all contained operations succeed.

A transaction might encapsulate a state change (unless it is a read-only one). A transaction must always leave the system in a consistent state, no matter how many concurrent transactions are interleaved at any given time.

# Consistency

Consistency means that constraints are enforced for every committed transaction. That implies that all Keys, Data types, Checks and Trigger are successful and no constraint violation is triggered.

# Isolation

Transactions require concurrency control mechanisms, and they guarantee correctness even when being interleaved. Isolation brings us the benefit of hiding uncommitted state changes from the outside world, as failing transactions shouldn’t ever corrupt the state of the system. Isolation is achieved through [concurrency control](http://en.wikipedia.org/wiki/Concurrency_control) using pessimistic or optimistic locking mechanisms.

# Durability

A successful transaction must permanently change the state of a system, and before ending it, the state changes are recorded in a persisted [transaction log](https://vladmihalcea.com/how-does-a-relational-database-work/). If our system is suddenly affected by a system crash or a power outage, then all unfinished committed transactions may be replayed.

For messaging systems like [JMS](http://en.wikipedia.org/wiki/Java_Message_Service), transactions are not mandatory. That’s why we have non-transacted [acknowledgement modes](http://docs.oracle.com/javaee/6/api/javax/jms/Session.html).

File system operations are usually non-managed, but if your business requirements demand transaction file operations, you might make use a tool such as [XADisk](https://xadisk.java.net/).

While messaging and file systems use transactions optionally, for database management systems transactions are compulsory.

# Challanges

ACID is old school. [Jim Gray](http://research.microsoft.com/en-us/um/people/gray/papers/theTransactionConcept.pdf) described atomicity, consistency and durability long before I was even born. But that particular paper doesn’t mention anything about isolation. This is understandable if we think of the production systems of the late 70’s, which according to Jim Gray:

“At present, the largest airlines and banks have about 10,000 terminals and about 100 active transactions at any instant”.

So all efforts were spent on delivering correctness rather than concurrency. Things have changed drastically ever since, and nowadays even modest set-ups are able to run 1000 TPS.

From a database perspective, the atomicity is a fixed property, but everything else may be traded off for performance/scalability reasons.

If the database system is composed of multiple nodes, then distributed system consistency (C in [CAP Theorem](https://en.wikipedia.org/wiki/CAP_theorem) not C in ACID) mandates that all changes be propagated to all nodes ([multi-master replication](http://en.wikipedia.org/wiki/Multi-master_replication)). If slaves nodes are updated asynchronously then we break the consistency rule, the system becoming “[eventually consistent](http://en.wikipedia.org/wiki/Eventual_consistency)“.

Peter Bailis has a [very good article](http://www.bailis.org/blog/linearizability-versus-serializability/) explaining the difference between Consistency in CAP Theorem and Consistency in ACID.

A transaction is a data state transition, so the system must operate as if all transactions occur in a serial form even if those are concurrently executed.  
If there would be only one connection running at all times, then serializability wouldn’t impose any concurrency control cost. In reality, all transactional systems must accommodate concurrent requests, hence serialization has its toll on scalability. The [Amdahl’s law](http://en.wikipedia.org/wiki/Amdahl%27s_law) describes the relation between serial execution and concurrency:

“The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.”

As you’ll see later, most database management systems choose (by default) to relax correctness guarantees to achieve better concurrency.

Playing with durability makes sense for [highly performing clustered databases](http://wiki.postgresql.org/images/3/3b/2011-11-11_PostgreSQL_SyncRepPerformance.pdf) if the enterprise system business requirements don’t mandate durable transactions. But, most often durability is better off untouched.

# Isolation Levels

Although some database management systems offer [MVCC](https://vladmihalcea.com/how-does-mvcc-multi-version-concurrency-control-work/), usually concurrency control is achieved through locking. But as we all know, locking increases the serializable portion of the executed code, affecting [parallelization](http://en.wikipedia.org/wiki/Amdahl%27s_law#Parallelization).

The SQL standard defines four Isolation levels:

* READ\_UNCOMMITTED
* READ\_COMMITTED
* REPEATABLE\_READ
* SERIALIZABLE

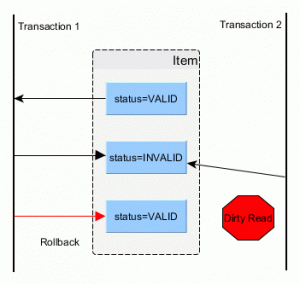
All but the SERIALIZABLE level are subject to data anomalies (phenomena) that might occur according to the following pattern:

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation Level** | **Dirty read** | **Non-repeatable read** | **Phantom read** |
| READ\_UNCOMMITTED | allowed | allowed | allowed |
| READ\_COMMITTED | prevented | allowed | allowed |
| REPEATABLE\_READ | prevented | prevented | allowed |
| SERIALIZABLE | prevented | prevented | prevented |

# Phenomena

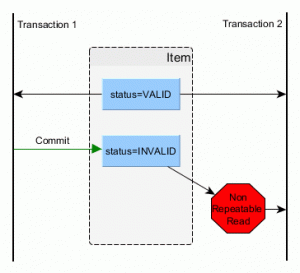
But what are all those phenomena we have just listed? Let’s discuss each and every one of them.

## Dirty read

[](https://vladmihalcea.files.wordpress.com/2014/01/acid-dirty-read.gif)

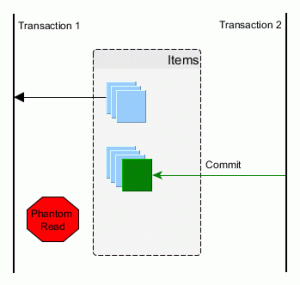
A dirty read happens when a transaction is allowed to read uncommitted changes of some other running transaction. This happens because there is no locking preventing it. In the picture above, you can see that the second transaction uses an inconsistent value as of the first transaction had rollbacked.

## Non-repeatable read

[](https://vladmihalcea.files.wordpress.com/2014/01/acid-non-repeatable-read.gif)

A non-repeatable read manifests when consecutive reads yield different results due to a concurring transaction that has just updated the record we’re reading. This is undesirable since we end up using stale data. This is prevented by holding a shared lock (read lock) on the read record for the whole duration of the current transaction.

## Phantom read

[](https://vladmihalcea.files.wordpress.com/2014/01/acid-phantom-read.gif)

A phantom read happens when a second transaction inserts a row that matches a previous select criteria of the first transaction. We, therefore, end up using stale data, which might affect our business operation. This is prevented using range locks or [predicate locking](https://vladmihalcea.com/how-does-database-pessimistic-locking-interact-with-insert-update-and-delete-sql-statements/).

## Even more ph

**54) Why the main method is static in java?**  
The method is static because otherwise there would be ambiguity: which constructor should be called? Especially if your class looks like this:

public class JavaClass

{

protected JavaClass(int x)

{ }

public void main(String[] args)

{

}

}

Should the JVM call new JavaClass(int)? What should it pass for x? If not, should the JVM instantiate JavaClass without running any constructor method? because that will special-case your entire class – sometimes you have an instance that hasn’t been initialized, and you have to check for it in every method that could be called. There are just too many edge cases and ambiguities for it to make sense for the JVM to have to instantiate a class before the entry point is called. That’s why main is static.

**What happens if you remove static modifier from the main method?**  
Program compiles successfully . But at runtime throws an error “NoSuchMethodError”.

**What is the** [**scope of variables**](https://www.geeksforgeeks.org/variable-scope-in-java/) **in Java in following cases?**

* **Member Variables** (Class Level Scope) : The member variables must be declared inside class (outside any function). They can be directly accessed anywhere in class
* **Local Variables** (Method Level Scope) : Variables declared inside a method have method level scope and can’t be accessed outside the method.
* **Loop Variables** (Block Scope) : A variable declared inside pair of brackets “{” and “}” in a method has scope withing the brackets only.

Read [more](https://www.geeksforgeeks.org/variable-scope-in-java/)

**What is** [**“this” keyword in java**](http://quiz.geeksforgeeks.org/this-reference-in-java/)**?**  
Within an instance method or a constructor, this is a reference to the current object — the object whose method or constructor is being called. You can refer to any member of the current object from within an instance method or a constructor by using this.  
Usage of this keyword

* Used to refer current class instance variable.
* To invoke current class constructor.
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* Used to return the current class instance.
* Used to invoke current class method (implicitly)

**What is an** [**abstract class**](https://www.geeksforgeeks.org/abstract-classes-in-java/)**? How abstract classes are similar or different in Java from C++?**  
Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

* Like C++, in Java, an instance of an abstract class cannot be created, we can have references of abstract class type though.
* Like C++, an abstract class can contain constructors in Java. And a constructor of abstract class is called when an instance of a inherited class is created
* In Java, we can have an abstract class without any abstract method. This allows us to create classes that cannot be instantiated, but can only be inherited.
* Abstract classes can also have final methods (methods that cannot be overridden). For example, the following program compiles and runs fine.

Read [more](https://www.geeksforgeeks.org/abstract-classes-in-java/)

**Which class is the superclass for every class ?**  
Object class

[**Can we overload main() method?**](https://www.geeksforgeeks.org/gfact-48-overloading-main-in-java/)  
The main method in Java is no extra-terrestrial method. Apart from the fact that main() is just like any other method & can be overloaded in a similar manner, JVM always looks for the method signature to launch the program.

* The normal main method acts as an entry point for the JVM to start the execution of program.
* We can overload the main method in Java. But the program doesn’t execute the overloaded main method when we run your program, we need to call the overloaded main method from the actual main method only.

Read [more](https://www.geeksforgeeks.org/gfact-48-overloading-main-in-java/)

**What is** [**object cloning**](http://quiz.geeksforgeeks.org/cloning-in-java/)**?**  
Object cloning means to create an exact copy of the original object. If a class needs to support cloning, it must implement java.lang.Cloneable interface and override clone() method from Object class. Syntax of the clone() method is :

protected Object clone() throws CloneNotSupportedException

If the object’s class doesn’t implement Cloneable interface then it throws an exception ‘CloneNotSupportedException’ .

Read [more](http://quiz.geeksforgeeks.org/cloning-in-java/)

**How is** [**inheritance in C++  different from Java?**](https://www.geeksforgeeks.org/comparison-of-inheritance-in-c-and-java/)

1. In Java, all classes inherit from the Object class directly or indirectly. Therefore, there is always a single inheritance tree of classes in Java, and Object class is root of the tree.
2. In Java, members of the grandparent class are not directly accessible. See [this G-Fact](https://www.geeksforgeeks.org/archives/15258) for more details.
3. The meaning of protected member access specifier is somewhat different in Java. In Java, protected members of a class “A” are accessible in other class “B” of same package, even if B doesn’t inherit from A (they both have to be in the same package).
4. Java uses *extends* keyword for inheritance. Unlike C++, Java doesn’t provide an inheritance specifier like public, protected or private. Therefore, we cannot change the protection level of members of base class in Java, if some data member is public or protected in base class then it remains public or protected in derived class. Like C++, private members of base class are not accessible in derived class.  
   Unlike C++, in Java, we don’t have to remember those rules of inheritance which are combination of base class access specifier and inheritance specifier.
5. In Java, methods are virtual by default. In C++, we explicitly use virtual keyword. See [this G-Fact](https://www.geeksforgeeks.org/archives/8876) for more details.
6. Java uses a separate keyword *interface* for interfaces, and *abstract* keyword for abstract classes and abstract functions.
7. Unlike C++, Java doesn’t support multiple inheritance. A class cannot inherit from more than one class. A class can implement multiple interfaces though.
8. In C++, default constructor of parent class is automatically called, but if we want to call parametrized constructor of a parent class, we must use [Initializer list](https://www.geeksforgeeks.org/archives/13797). Like C++, default constructor of the parent class is automatically called in Java, but if we want to call parameterized constructor then we must use super to call the parent constructor.

See examples [here](https://www.geeksforgeeks.org/comparison-of-inheritance-in-c-and-java/)

**Why method overloading is not possible by changing the return type in java?**  
In C++ and Java, functions can not be overloaded if they differ only in the return type . The return type of functions is not a part of the mangled name which is generated by the compiler for uniquely identifying each function. The No of arguments, Type of arguments & Sequence of arguments are the parameters which are used to generate the unique mangled name for each function. It is on the basis of these unique mangled names that compiler can understand which function to call even if the names are same(overloading).

**Can we override private methods in Java?**  
No, a private method cannot be overridden since it is not visible from any other class. Read [more](https://www.geeksforgeeks.org/can-override-private-methods-java/)

**What is** [**blank final variable**](https://www.geeksforgeeks.org/blank-final-in-java/)**?**  
A final variable in Java can be assigned a value only once, we can assign a value either in declaration or later.

final int i = 10;

i = 30; // Error because i is final.

A **blank final** variable in Java is a [final](https://www.geeksforgeeks.org/g-fact-48/) variable that is not initialized during declaration. Below is a simple example of blank final.

// A simple blank final example

final int i;

i = 30;

Read [more](https://www.geeksforgeeks.org/blank-final-in-java/)

**What is** [**“super” keyword in java**](http://quiz.geeksforgeeks.org/super-keyword/)**?**  
The super keyword in java is a reference variable that is used to refer parent class objects. The keyword “super” came into the picture with the concept of Inheritance. Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.  
Various scenarios of using java super Keyword:

* super is used to refer immediate parent instance variable
* super is used to call parent class method
* super() is used to call immediate parent constructor

Read [more](http://quiz.geeksforgeeks.org/super-keyword/)

**What is** [**static variable in Java**](https://www.geeksforgeeks.org/static-class-in-java/)**?**  
The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

* variable (also known as class variable)
* method (also known as class method)
* block
* nested class

**Differences between** [**HashMap and HashTable in Java**](http://quiz.geeksforgeeks.org/differences-between-hashmap-and-hashtable-in-java/)**.**  
1. HashMap is non synchronized. It is not-thread safe and can’t be shared between many threads without proper synchronization code whereas Hashtable is synchronized. It is thread-safe and can be shared with many threads.  
2. HashMap allows one null key and multiple null values whereas Hashtable doesn’t allow any null key or value.  
3. HashMap is generally preferred over HashTable if thread synchronization is not needed  
[Read more](http://quiz.geeksforgeeks.org/differences-between-hashmap-and-hashtable-in-java/)

**How are Java** [**objects stored in memory**](https://www.geeksforgeeks.org/g-fact-46/)**?**  
In Java, all objects are dynamically allocated on **Heap**. This is different from C++ where objects can be allocated memory either on Stack or on Heap. In C++, when we allocate abject using new(), the object is allocated on Heap, otherwise on Stack if not global or static.  
In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on heap.  Read [more](https://www.geeksforgeeks.org/g-fact-46/)

