# Android studio guide

# The mystery: Android Studio's Project Structure and Build System

I don't know if this is because of the Gradle Build System (I'd wager it is), but I'll tell you what I've understood so far.

**Update 4:** 2014/09/11 Added **Cheat Sheet** for BuildTypes, Flavors and Variants(I finally feel confident to write this :D)  
**Update 3:** 2014/09/11 Updated the comparison workspaces and projects to be precise  
**Update 2:** 2014/04/17 Added more detail to AS project structure  
**Update 1:** 2013/07/29 Added IntelliJ Project Structure

The IntelliJ's Project structure (shown at the end) is for IntelliJ with the android plugin. The Android Studio, however, has a project structure divided like so:

## **Structure: Projects and Modules**

**module** in Android Studio is like a **project** in Eclipse

**project** in Android Studio is like a **workspace** in Eclipse (to be precise, a workspace with interdependent projects)

From [the documentation](http://confluence.jetbrains.com/display/IDEADEV/Structure+of+IntelliJ+IDEA+Project) (Android Studio is based on Intellij IDEA) :

Whatever you do in IntelliJ IDEA, you do that in the context of a project. A project is an organizational unit that represents a complete software solution.

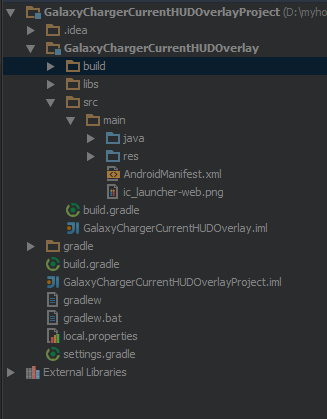
Your finished product may be decomposed into a series of discrete, isolated modules, but it's a project definition that brings them together and ties them into a greater whole.

For Android, it means one project per app, and one module per library and per test app.

There are multiple issues if you try to build multiple apps within the same project. It's possible, but if you try (like I did), you will see that almost everything is designed to work with a single app per project.

For example, there is an option to "rebuild the project", which makes no sense with multiple apps, many other project settings would be useless, and the built-in VCS system isn't great when you have multiple repositories.

## **Structure: Folder Structure**



### **Top Level Folders**

### **1. Main Project**

This would be entire **project context** (**Eclipse Land:** Like your workspace but limited to what's relevant to your project). Ex: HelloWorldProject if the name of the application you gave was HelloWorld

### **2. .idea**

This where project specific metadata is stored by Android Studio (AS). (**Eclipse Land:** project.properties file)

### **3. Project Module**

This is the actual project. ex: HelloWorld if your application name you gave was HelloWorld

### **4. gradle**

This is where the gradle build system's jar wrapper i.e. this jar is how AS communicates with gradle installed in Windows (the OS in my case).

### **5. External Libraries**

This is not actually a folder but a place where Referenced Libraries (**Eclipse Land:** Referenced Libraries) are shown. Here's where the Targeted Platform is shown etc.

[**Side note:** This where many of us in Eclipse Land used to delete the referenced libraries and Fix Project Properties to fix reference errors, remember?]

## **Project Folder in Detail**

This is number #3 in the above list. Has the following sub dirs

### **1. build**

This has all the complete output of the make process i.e. classes.dex, compiled classes and resources, etc.

In the Android Studio GUI, only a few folders are shown. The important part is that **your R.java is found here** under **build/source/<flavor>/r/<build type(optional)>/<package>/R.java**

### **2. libs**

This is the standard libs folder that you see in **eclipse land** too

### **3. src**

Here, you only see the java and res folder which correspond to the src folder and res folder in**Eclipse Land**. This is much welcomed simplification IMHO.

## **Note on Modules:**

Modules are like **Eclipse Land** projects. Here the idea is that you have one application project (Module #3 in the list above) and several library projects(as separate Modules under the global project folder (#1 in the above list)) which the application project depends on. How these library projects can be re-used in other applications, I still haven't found out.

[**Side note:** The whole re-organization has some benefits like simplifications in src folder, but so many complications. The complications are mainly due **VERY VERY** thin documentation on this new project layout.]

# The New Build System

[User Guide for the new Build System](http://tools.android.com/tech-docs/new-build-system/user-guide)

## **Explanation of flavors and buildTypes, etc - What is the hullabaloo about?**

### **CheatSheet for flavors and buildTypes**

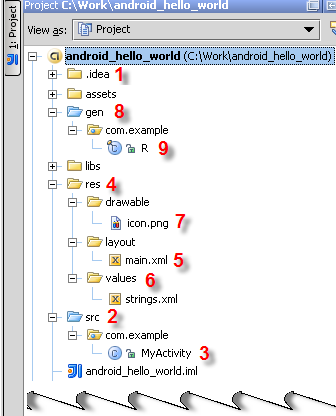
**BuildType:** debug and release are buildTypes available by default on all projects. They are for building/compiling the **SAME CODE** to generate different APKs. For example on release APKs you would want to run proguard(for obfuscation), sign it with your key (as against the debug key), run optimizations (maybe via proguard or other tools), use slightly different packageNames (we use com.company.product for release and com.company.product.debug for debug), etc. We also use a debug flag (BuildConfig.DEBUG) to turn off logging to logcat (since it makes the app slow) on release builds. This makes for a faster debug build during development but also an optimized release build to put on play store.

**Product Flavor:** There are no default flavors available (or to be precise, the default flavor is blank/nameless). Flavors could be **free version** or **paid version** where they have **DIFFERENT CODE**. They share the same Main Code but different versions(or no versions) of a few source code files or resources.

**BuildVariant:** A buildVariant is what a generated APK actually corresponds to. They are named like so (in order) **Product Flavor + Build Type = Build Variant**.  
**Example 1:** if you have free and paid as two flavors. The build variants you would get are:  
Free - debug  
Free - release  
Paid - debug  
Paid - release  
So that is 4 possible APK configurations. A few configurations may not make sense in a particular project, but they **are** available.

**Example 2:** (for new projects/ no flavors) You have 2 buildVariants or APKs available, since the default flavor is nameless/blank:  
debug  
release

# Compare this with [Intellij's Project Structure](http://wiki.jetbrains.net/intellij/Developing_applications_for_Android_in_IntelliJ_IDEA#Creating_a_New_Project) if that helps:



**The .idea (1)** folder contains a number of subfolders, mainly with internal IntelliJ IDEA information.

**The src (2)** folder contains the MyActivity.java **(3) file source code** that implements the functionality of your application. The file belongs to the com.example package.

**The res (4)** folder contains various visual resources.

**The layout/main.xml file (5)** defines the appearance of the application constituted of resources of various types.

**The values folder (6)** is intended for storing .xml files that describe resources of various types. Presently, the folder contains a strings.xml file with String resources definitions. As you will see from the Adding a Color section, the layout folder can also contain, for example, a descriptor of colors.

**The drawable folder (7)** contains images.

**The gen (8) folder** contains the **R.java (9)** file that links the visual resources and the Java source code. As you will see from the sections below, IntelliJ IDEA supports tight integration between static resources and R.java. As soon as any resources are added or removed, the corresponding classes and class fields in R.java are automatically generated or removed accordingly. The R.java file also belongs to the com.example package.

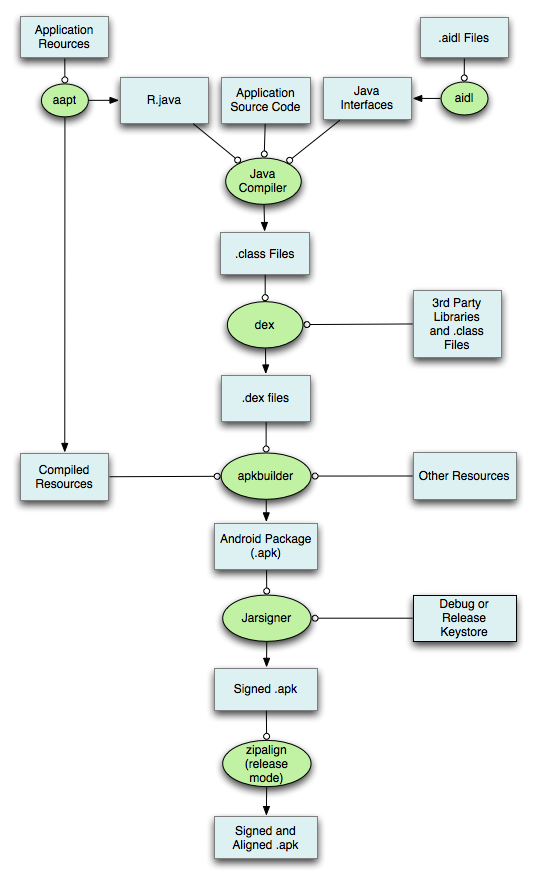
# New build system guide

<http://tools.android.com/tech-docs/new-build-system/user-guide>

<http://tools.android.com/tech-docs/new-build-system/tips>

# Build system overview

<http://developer.android.com/sdk/installing/studio-build.html>



# Android studio Tips & Tricks

<http://developer.android.com/sdk/installing/studio-tips.html>

## **Moving Around**

#### Opening Class/File/Symbol

* Open class

cmd+o || ctrl+n

Imagine that you must go to a class named “MainActivity”, just use this shortcut and start typing “MainA”.

* Open File

cmd+shift+o || ctrl+shift+n

Works like the “Open Class” shortcut but on every files in your project. It is very useful to open your AndroidManifest.xml or anything that sits in the res/ or assets/ folder.

* Open Symbol

cmd+alt+o || alt+shift+n

A very powerful but little less known variation of the previous tips: You can use this to go directly to a method or field by searching its name!

For example, if you know that you have a method named getFormattedDate() somewhere in your project, you can type it in the Open Symbol dialog to go directly to it.

* Tips
* Partial Matching
* You can enter incomplete strings and it will work. For example. if you are searching for a class named “ItemDetailFragment”, you can actually type “IDF” and it will find it.
* Line Number
* Imagine that your colleague just told you that the juicy part is in ExcitingClass at line 23. You can open the file directly by appending a “:” to the class name in the Open Class dialog. e.g.:
* ExcitingClass:22

You can also combine it with partial matching and type something like:

EC:22

# Training android step by step

## Building your first app

## Send an simple request

At a high level, you use Volley by creating a RequestQueue and passing it Request objects. The RequestQueue manages worker threads for running the network operations, reading from and writing to the cache, and parsing responses. Requests do the parsing of raw responses and Volley takes care of dispatching the parsed response back to the main thread for delivery.

This lesson describes how to send a request using the Volley.newRequestQueue convenience method, which sets up a RequestQueue for you. See the next lesson, Setting Up a RequestQueue, for information on how to set up a RequestQueue yourself.

This lesson also describes how to add a request to a RequestQueue and cancel a request.

**Add the INTERNET Permission**

To use Volley, you must add the [android.permission.INTERNET](https://developer.android.com/reference/android/Manifest.permission.html#INTERNET) permission to your app's manifest. Without this, your app won't be able to connect to the network.

**Use newRequestQueue**

Volley provides a convenience method Volley.newRequestQueue that sets up a RequestQueue for you, using default values, and starts the queue. For example:

final TextView mTextView = (TextView) findViewById(R.id.text);  
...  
  
// Instantiate the RequestQueue.  
RequestQueue queue = Volley.newRequestQueue(this);  
String url ="http://www.google.com";  
  
// Request a string response from the provided URL.  
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,  
            new Response.Listener<String>() {  
    @Override  
    public void onResponse(String response) {  
        // Display the first 500 characters of the response string.  
        mTextView.setText("Response is: "+ response.substring(0,500));  
    }  
}, new Response.ErrorListener() {  
    @Override  
    public void onErrorResponse(VolleyError error) {  
        mTextView.setText("That didn't work!");  
    }  
});  
// Add the request to the RequestQueue.  
queue.add(stringRequest);

Volley always delivers parsed responses on the main thread. Running on the main thread is convenient for populating UI controls with received data, as you can freely modify UI controls directly from your response handler, but it's especially critical to many of the important semantics provided by the library, particularly related to canceling requests.

See [Setting Up a RequestQueue](https://developer.android.com/training/volley/requestqueue.html) for a description of how to set up a RequestQueue yourself, instead of using theVolley.newRequestQueue convenience method.

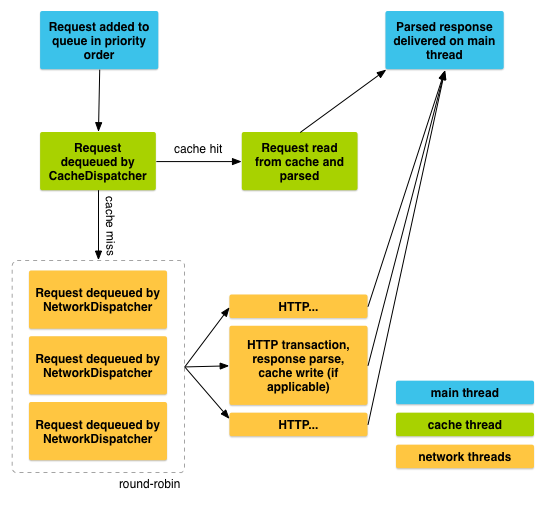
**Send a Request**

To send a request, you simply construct one and add it to the RequestQueue with add(), as shown above. Once you add the request it moves through the pipeline, gets serviced, and has its raw response parsed and delivered.

When you call add(), Volley runs one cache processing thread and a pool of network dispatch threads. When you add a request to the queue, it is picked up by the cache thread and triaged: if the request can be serviced from cache, the cached response is parsed on the cache thread and the parsed response is delivered on the main thread. If the request cannot be serviced from cache, it is placed on the network queue. The first available network thread takes the request from the queue, performs the HTTP transaction, parsse the response on the worker thread, writes the response to cache, and posts the parsed response back to the main thread for delivery.

Note that expensive operations like blocking I/O and parsing/decoding are done on worker threads. You can add a request from any thread, but responses are always delivered on the main thread.

Figure 1 illustrates the life of a request:



To cancel a request, call cancel() on your Request object. Once cancelled, Volley guarantees that your response handler will never be called. What this means in practice is that you can cancel all of your pending requests in your activity's [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()) method and you don't have to litter your response handlers with checks forgetActivity() == null, whether onSaveInstanceState() has been called already, or other defensive boilerplate.

To take advantage of this behavior, you would typically have to track all in-flight requests in order to be able to cancel them at the appropriate time. There is an easier way: you can associate a tag object with each request. You can then use this tag to provide a scope of requests to cancel. For example, you can tag all of your requests with the [Activity](https://developer.android.com/reference/android/app/Activity.html) they are being made on behalf of, and call requestQueue.cancelAll(this) from [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()). Similarly, you could tag all thumbnail image requests in a [ViewPager](https://developer.android.com/reference/android/support/v4/view/ViewPager.html) tab with their respective tabs and cancel on swipe to make sure that the new tab isn't being held up by requests from another one.

Here is an example that uses a string value for the tag:

1. Define your tag and add it to your requests.

public static final String TAG = "MyTag";  
StringRequest stringRequest; // Assume this exists.  
RequestQueue mRequestQueue;  // Assume this exists.  
  
// Set the tag on the request.  
stringRequest.setTag(TAG);  
  
// Add the request to the RequestQueue.  
mRequestQueue.add(stringRequest);

1. In your activity's [onStop()](https://developer.android.com/reference/android/app/Activity.html#onStop()) method, cancel all requests that have this tag.

@Override  
protected void onStop () {  
    super.onStop();  
    if (mRequestQueue != null) {  
        mRequestQueue.cancelAll(TAG);  
    }  
}

Take care when canceling requests. If you are depending on your response handler to advance a state or kick off another process, you need to account for this. Again, the response handler will not be called.

# Parcelable vs Serializable

When starting on Android, we all learn that we cannot just pass object references to activities and fragments, we have to put those in an Intent / Bundle.

Looking at the api, we realize that we have two options, we can either make our objects Parcelable or Serializable. As Java developers, we already know of the Serializable mechanism, so why bother with Parcelable?

To answer this, lets take a look at both approaches.

**Serializable, the Master of Simplicity**

*// access modifiers, accessors and constructors omitted for brevity*

**public** **class** **SerializableDeveloper** **implements** Serializable

String name**;**

**int** yearsOfExperience**;**

List**<**Skill**>** skillSet**;**

**float** favoriteFloat**;**

**static** **class** **Skill** **implements** Serializable **{**

String name**;**

**boolean** programmingRelated**;**

**}**

**}**

The beauty of serializable is that you only need to implement the Serializable interface on a class and its children. It is a marker interface, meaning that there is no method to implement, Java will simply do its best effort to serialize it efficiently.

The problem with this approach is that reflection is used and it is a slow process. This mechanism also tends to create a lot of temporary objects and cause quite a bit of garbage collection.

**Parcelable, the Speed King**

**class** **ParcelableDeveloper** **implements** Parcelable **{**

String name**;**

**int** yearsOfExperience**;**

List**<**Skill**>** skillSet**;**

**float** favoriteFloat**;**

ParcelableDeveloper**(**Parcel in**)** **{**

**this.**name **=** in**.**readString**();**

**this.**yearsOfExperience **=** in**.**readInt**();**

**this.**skillSet **=** **new** ArrayList**<**Skill**>();**

in**.**readTypedList**(**skillSet**,** Skill**.**CREATOR**);**

**this.**favoriteFloat **=** in**.**readFloat**();**

**}**

**void** **writeToParcel(**Parcel dest**,** **int** flags**)** **{**

dest**.**writeString**(**name**);**

dest**.**writeInt**(**yearsOfExperience**);**

dest**.**writeTypedList**(**skillSet**);**

dest**.**writeFloat**(**favoriteFloat**);**

**}**

**int** **describeContents()** **{**

**return** 0**;**

**}**

**static** **final** Parcelable**.**Creator**<**ParcelableDeveloper**>** CREATOR

**=** **new** Parcelable**.**Creator**<**ParcelableDeveloper**>()** **{**

ParcelableDeveloper **createFromParcel(**Parcel in**)** **{**

**return** **new** **ParcelableDeveloper(**in**);**

**}**

ParcelableDeveloper**[]** **newArray(int** size**)** **{**

**return** **new** ParcelableDeveloper**[**size**];**

**}**

**};**

**static** **class** **Skill** **implements** Parcelable **{**

String name**;**

**boolean** programmingRelated**;**

Skill**(**Parcel in**)** **{**

**this.**name **=** in**.**readString**();**

**this.**programmingRelated **=** **(**in**.**readInt**()** **==** 1**);**

**}**

@Override

**void** **writeToParcel(**Parcel dest**,** **int** flags**)** **{**

dest**.**writeString**(**name**);**

dest**.**writeInt**(**programmingRelated **?** 1 **:** 0**);**

**}**

**static** **final** Parcelable**.**Creator**<**Skill**>** CREATOR

**=** **new** Parcelable**.**Creator**<**Skill**>()** **{**

Skill **createFromParcel(**Parcel in**)** **{**

**return** **new** **Skill(**in**);**

**}**

Skill**[]** **newArray(int** size**)** **{**

**return** **new** Skill**[**size**];**

**}**

**};**

@Override

**int** **describeContents()** **{**

**return** 0**;**

**}**

**}**

**}**

According to google engineers, this code will run significantly faster. One of the reasons for this is that we are being explicit about the serialization process instead of using reflection to infer it. It also stands to reason that the code has been heavily optimized for this purpose.

However, it is obvious here that implementing Parcelable is not free. There is a significant amount of boilerplate code and it makes the classes harder to read and maintain.

**Speed Tests**

Of course, we want to know how much faster Parcelable is.

The methodology

Mimic the process of passing object to an activity by putting an object in a bundle and calling Bundle#writeToParcel(Parcel, int) and then fetching it back

Run this in a loop 1000 times

Do an average on 10 separate runs to account for memory allocation, other apps using the cpu, etc

The object under test are the SerializableDeveloper and the ParcelableDeveloper shown above

Test on multiple devices - android versions

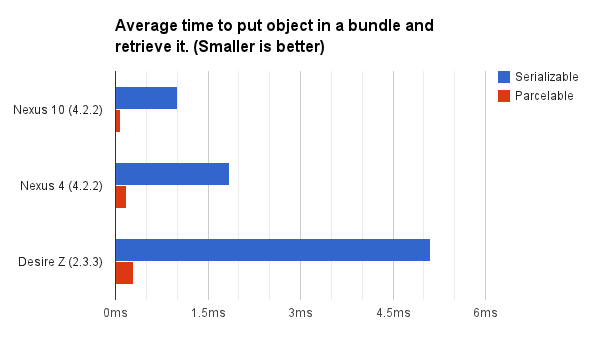
LG Nexus 4 - Android 4.2.2

Samsung Nexus 10 - Android 4.2.2

HTC Desire Z - Android 2.3.3

The results

### **The results**

[](http://www.developerphil.com/assets/parcelable-vs-serializable.png)

Nexus 10

Serializable: 1.0004ms, Parcelable: 0.0850ms - 10.16x improvement.

Nexus 4

Serializable: 1.8539ms - Parcelable: 0.1824ms - 11.80x improvement.

Desire Z

Serializable: 5.1224ms - Parcelable: 0.2938ms - 17.36x improvement.

There you have it: Parcelable is more than 10x faster than Serializable! It is also interesting to note that even on a Nexus 10, a pretty simple object can take about 1 millisecond to go through a full serialize/deserialize cycle.

**The Bottom Line**

If you want to be a good citizen, take the extra time to implement Parcelable since it will perform 10 times faster and use less resources.

However, in most cases, the slowness of Serializable won’t be noticeable. Feel free to use it but remember that serialization is an expensive operation so keep it to a minimum.

If you are trying to pass a list with thousands of serialized objects, it is possible that the whole process will take more than a second. It can make transitions or rotation from portrait to landscape feel very sluggish.

Understand the serialVersionUID

If you have ever implemented **Serializable**interface, you must encounter this warning message

The serializable class xxx does not **declare** a static final serialVersionUID field of **type** long

So…what is serialVersionUID?

The serialVersionUID is used as a version control in a Serializable class. If you do not explicitly declare a serialVersionUID, JVM will do it for you automatically, based on various aspects of your Serializable class, as described in the [**Java(TM) Object Serialization Specification**](http://docs.oracle.com/javase/1.5.0/docs/api/java/io/Serializable.html).

1. SerialVersionUID Example

The above statement is a bit hard to understand at the beginning (at least I did), let start an example to understand how Serializable class use SerialVersionUID to implement version control.

1.1 Address.java

A serializable class with a serialVersionUID of 1L.

**import** java.io.Serializable;

**public** **class** Address **implements** Serializable{

**private** **static** **final** **long** serialVersionUID = 1L;

String street;

String country;

**public** **void** setStreet(String street){

**this**.street = street;

}

**public** **void** setCountry(String country){

**this**.country = country;

}

**public** String getStreet(){

**return** **this**.street;

}

**public** String getCountry(){

**return** **this**.country;

}

@Override

**public** String toString() {

**return** **new** StringBuffer(" Street : ")

.append(**this**.street)

.append(" Country : ")

.append(**this**.country).toString();

}

}

1.2 WriteObject.java

A simple class to write / serialize the Address object into a file – “c:\\address.ser”.

**import** java.io.FileOutputStream;

**import** java.io.ObjectOutputStream;

**public** **class** WriteObject{

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

Address address = **new** Address();

address.setStreet("wall street");

address.setCountry("united states");

**try**{

FileOutputStream fout = **new** FileOutputStream("c:**\\**address.ser");

ObjectOutputStream oos = **new** ObjectOutputStream(fout);

oos.writeObject(address);

oos.close();

System.out.println("Done");

}**catch**(Exception ex){

ex.printStackTrace();

}

}

}

1.3 ReadObject.java

A simple class to read / deserialize the Address object from file – “c:\\address.ser”.

**import** java.io.FileInputStream;

**import** java.io.ObjectInputStream;

**public** **class** ReadObject{

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

Address address;

**try**{

FileInputStream fin = **new** FileInputStream("c:**\\**address.ser");

ObjectInputStream ois = **new** ObjectInputStream(fin);

address = (Address) ois.readObject();

ois.close();

System.out.println(address);

}**catch**(Exception ex){

ex.printStackTrace();

}

}

}

2. Testing

Let do some testing to demonstrate the use of serialVersionUID.

2.1 Same serialVersionUID

Same serialVersionUID , there is no problem during the deserialization process

javac Address.java

javac WriteObject.java

javac ReadObject.java

java WriteObject

java ReadObject

Street : **wall** street Country : united states

2.2 Different serialVersionUID

In Address.java, **change the serialVersionUID to 2L** (it was 1L), and compile it again.

javac Address.java

java ReadObject

java.io.InvalidClassException: Address; **local** class incompatible:

stream classdesc serialVersionUID = 1, **local** class serialVersionUID = 2

...

at ReadObject.main**(**ReadObject.java:14**)**

The “InvalidClassException” will raise, because you write a serialization class with **serialVersionUID “1L”** but try to retrieve it back with updated serialization class, **serialVersionUID “2L”**.

The serialVersionUID have to match during the serialization and deserialization process.

**When should update your serialVersionUID?**  
When your serialization class is updated with some incompatible Java type changes to a serializable class, you have to update your serialVersionUID.

For detail about the compatible and incompatible Java type changes to a serializable class, see the[**Java Object Serialization Specification.**](http://docs.oracle.com/javase/6/docs/platform/serialization/spec/serialTOC.html)

3. What’s wrong with the default serialVersionUID?

If no serialVersionUID is declared, JVM will use its own algorithm to generate a default SerialVersionUID, you can check the algorithm [**here**](http://docs.oracle.com/javase/6/docs/platform/serialization/spec/class.html#4100).

**The default serialVersionUID computation is highly sensitive to class details and may vary from different JVM implementation**, and result in an unexpected InvalidClassExceptions during the deserialization process.

3.1 Client / Server environment

- Client is using SUN’s JVM in Windows.– Server is using JRockit in Linux.

The client sends a serializable class with default generated serialVersionUID (e.g 123L) to the server over socket, the server may generate a different serialVersionUID (e.g 124L) during deserialization process, and raises an unexpected InvalidClassExceptions.

3.2 File / Database environment

- App #1 is using SUN’s JVM in Windows.– App #2 is using JRockit in Linux.

Serialization has allowed to save into a file or database. App #1 stores a serializable class into database by default generated serialVersionUID (e.g 123L), while App #2 may generate a different serialVersionUID (e.g 124L) during deserialization process, and raise an unexpected InvalidClassExceptions.

You can check here for the [**List of the JVM implementation**](http://en.wikipedia.org/wiki/List_of_JVM_implementations).

4. How to generate serialVersionUID

You can use JDK “serialver” or Eclipse IDE to generate serialVersionUID automatically, [**see detail**](http://www.mkyong.com/java/how-to-generate-serialversionuid/).

Conclusion

SUN is highly recommended developers to declare the serialVersionUID in order to avoid the different JVM issue listed above, however I rather recommend you should understand what is serialization, how serialVersionUID implement version control and why your class need to use serialization. Understand the serialVersionUID concept is better than blindfold to any recommendation.

# Android application – performance and more

* StrictMode
* **package** de.vogella.android.strictmode;
* **import** java.io.BufferedWriter;
* **import** java.io.OutputStreamWriter;
* **import** android.app.Activity;
* **import** android.os.Bundle;
* **import** android.os.StrictMode;
* **public** **class** TestStrictMode **extends** Activity {
* *@Override*
* **public** **void** onCreate(Bundle savedInstanceState) {
* **super**.onCreate(savedInstanceState);
* *// Activate StrictMode*
* StrictMode.setThreadPolicy(**new** StrictMode.ThreadPolicy.Builder()
* .detectAll()
* .detectDiskReads()
* .detectDiskWrites()
* .detectNetwork()
* *// alternatively .detectAll() for all detectable problems*
* .penaltyLog()
* .penaltyDeath()
* .build());
* StrictMode.setVmPolicy(**new** StrictMode.VmPolicy.Builder()
* .detectLeakedSqlLiteObjects()
* .detectLeakedClosableObjects()
* *// alternatively .detectAll() for all detectable problems*
* .penaltyLog()
* .penaltyDeath()
* .build());
* *// Test code*
* setContentView(R.layout.main);
* **super**.onCreate(savedInstanceState);
* setContentView(R.layout.main);
* String eol = System.getProperty("line.separator");
* **try** {
* BufferedWriter writer =
* **new** BufferedWriter(**new** OutputStreamWriter(openFileOutput("myfile",
* MODE\_WORLD\_WRITEABLE)));
* writer.write("This is a test1." + eol);
* writer.write("This is a test2." + eol);
* writer.write("This is a test3." + eol);
* writer.write("This is a test4." + eol);
* writer.write("This is a test5." + eol);
* writer.close();
* } **catch** (Exception e) {
* e.printStackTrace();
* }
* }
* }
* Developer Settings
* Lint

The Android lint tool is a static code analysis tool for Android applications. It provides command line and Eclipse supported checks for potential bugs and optimization improvements for Android applications.

Some checks are performed by default in Eclipse. To run active Lint checks in Eclipse on your Android application, right-click on your project and select Android Tools → Run Lint: Check for Common Errors.

You see the generated warnings in the Lint Warning view .

In the Eclipse Preferences you can configure the Lint checks under Window → Preferences → Android → Lint Error Checking.

You can run link also from the command line.

# run lint check **for** myproject

lint myproject

* Traceview

android.os.Debug.startMethodTracing("yourstring");

// ... your code is here

android.os.Debug.stopMethodTracing();

adb pull /sdcard/yourstring.trace

traceview yourstring

* Hierarchy Viewer
* Layout optimization
* Memory Dumps
* Systrace

Systrace allows to measure the performance directly at kernel level. To enable systrace, select the Developer options and select the Enable traces entry. In the next dialog you can define what type of events should be profiled, e.g. Graphics and View.

To use systrace, open a terminal and run systrace.py from the android\_sdk\_installdir/tools/systrace directory. You may have to set it to executable (chmod 777 systrace.py on Linux).

You can also start Systrace directly from Eclipse via the DDMS perspective.

Systrace captures events for 5 seconds. As result Systrace creates a HTML file which allows you to analyze potential issues.

* Pixel Perfect perspective

Not performance related but still useful is the Pixel Perfect perspective which allow to display the screen of the currently running application on the emulator or device and magnify the image of that screen.

* Simulating pixel density

You can use the command line to simulate different device densities and display resolutions.

This allows you to use a device with a high device density and resolution to simulate other devices.

// Set the display size

adb shell am display-size 600x800

// Set the display density

adb shell am display-density 80

* Android templates

You can define your own templates for the Android project creation wizard. See the following links for more information.

Roman Nurik: https://plus.google.com/113735310430199015092/posts/XTKTamk4As8

Official documentation: https://dl.dropbox.com/u/231329/android/templatedocs/index.html

Additional code templates - <https://github.com/jgilfelt/android-adt-templates>

* Profile GPU rendering

In the Developer options in the Setting or your Android device you can activate Profile GPU rendering. With this option the system tracks the time it took to draw the last 128 frames.

After activating this and restarting your application you can get the information via the following command.

adb shell dumpsys gfxinfo your\_package

To measure the frame rate of your application, e.g. by scrolling on a list view. In most cases you need to interact with your application to trigger that it re-draws itself.

In the resulting log, look for a section called Profile data in ms.

* Analyzing Overdraw

Overdraw happens if you draw something on top of something else. For example an activity has a Window as background. If a TextView it added to the application, the TextView is drawn on top of the Window.

Overdrawn is therefore intended to happen. But unnecessary overdraw should be avoid to have the best performance possible.

Unnecessary overdraw can be caused by complex view hierarchies. In general a 2x overdraw (a pixel gets drawn three times) is standard and expected but more should be avoided.

You can enable a visualization of overdraw via the Development Settings with the Show GPU overdraw setting. This section adds colors to your screen based on the number of overdraws. The following table explains the used color schema.

| **Color** | **Meaning** |
| --- | --- |
| No color | No overdraw |
| Blue | 1x overdraw, pixel was painted twice |
| Green | 2x |
| Light red | 3x, might indicate a problem, small red areas are still ok |
| Dark red | 4x, pixel was pained 5 times or more, indicates a problem. |

# Gang of Four (GoF) Design Pattern

Gang of Four (GoF) Patterns are 23 main software design patterns providing recurring solutions to common problems in software design. They were developed by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, often referred to as the Gang of Four.

We have started this series to explain each of these patterns with a real world example and how to develop and use them in Java.

Creational Design Pattern:

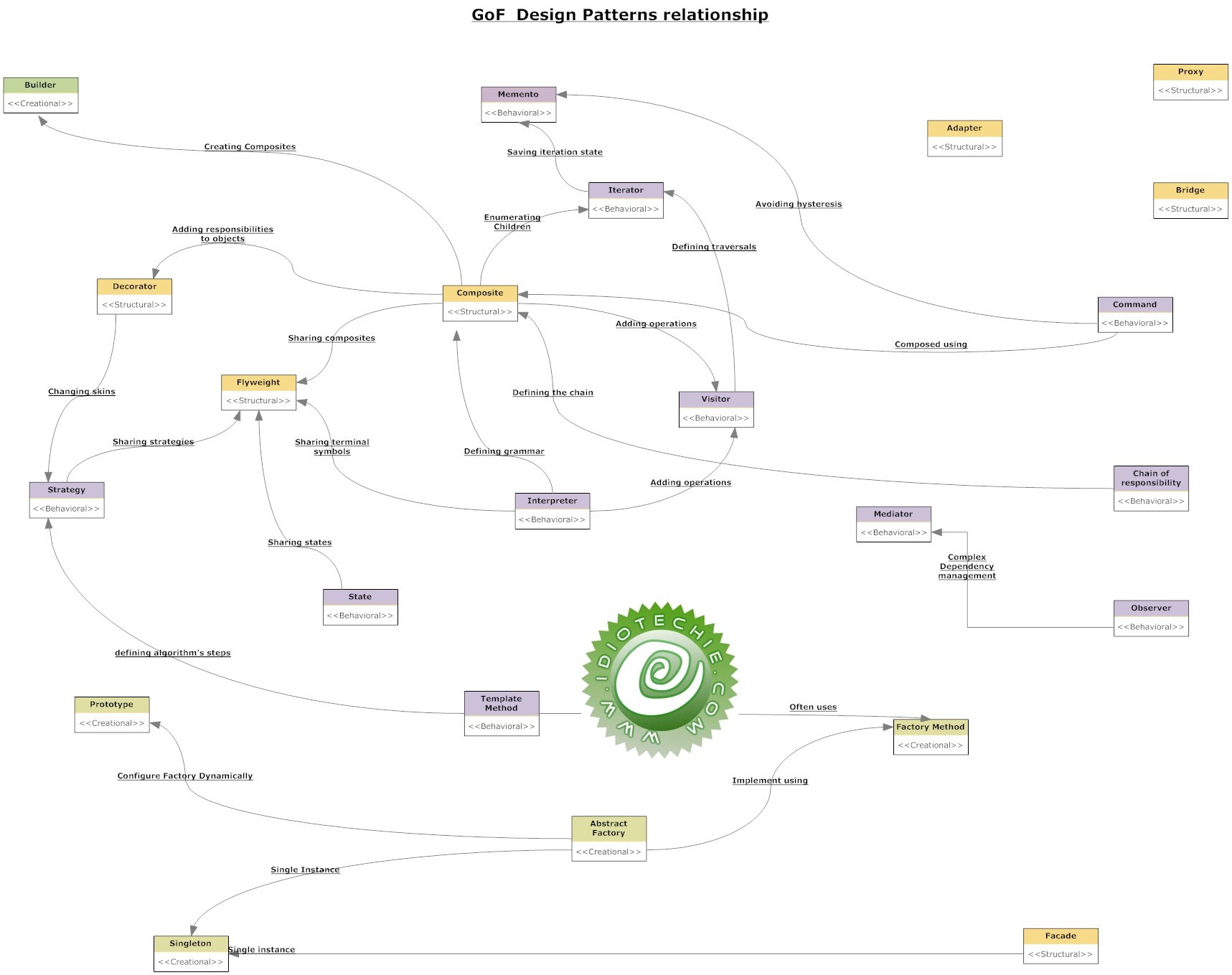
This is all about object creation in a form that they can follow the architecture principle of loose coupling, encapsulation and abstraction. This pattern can provide a great deal of flexibility about which objects are created, how those objects are created, and how they’re initialized. And the most important thing is that the client remains completely unaware of overall object creation process.

Structural Design Pattern:

This pattern is used to form larger object structures from many different objects. They show you how to use composition to join different pieces of a system together in a flexible and extensible fashion. This pattern help you guarantee that when one of the parts changes, the entire structure doesn’t need to change. They also show you how to recast pieces that don’t fit into pieces that do fit.

Behaviour Design Pattern:

This pattern is used to manage relationships, interaction, algorithms and responsibilities between various objects. It also follows the abstraction principle whereby it abstracts the action. This pattern focuses on the interaction between the cooperating objects in a manner that the objects are communicating while maintaining loose coupling.



**Types of Creational Design Patterns** (as defined by Wikipedia):

**Abstract Factory** groups object factories that have a common theme.

**Builder** constructs complex objects by separating construction and representation.

**Factory Method** creates objects without specifying the exact class to create.

**Prototype** creates objects by cloning an existing object.

**Singleton** restricts object creation for a class to only one instance.

**Types of Structural Design Pattern** (as defined by Wikipedia):

**Adapter** allows classes with incompatible interfaces to work together by wrapping its own interface around that of an already existing class.

**Bridge** decouples an abstraction from its implementation so that the two can vary independently.

**Composite** composes zero-or-more similar objects so that they can be manipulated as one object.

**Decorator** dynamically adds/overrides behaviour in an existing method of an object.

**Facade** provides a simplified interface to a large body of code.

**Flyweight** reduces the cost of creating and manipulating a large number of similar objects.

**Proxy** provides a placeholder for another object to control access, reduce cost, and reduce complexity.

**Types of Behavioral Design Pattern** (as defined by Wikipedia):

**Command** creates objects which encapsulate actions and parameters.

**Interpreter** implements a specialized language.

**Iterator** accesses the elements of an object sequentially without exposing its underlying representation.

**Mediator** allows loose coupling between classes by being the only class that has detailed knowledge of their methods.

**Memento** provides the ability to restore an object to its previous state (undo).

**Observer** is a publish/subscribe pattern which allows a number of observer objects to see an event.

**State** allows an object to alter its behavior when its internal state changes.

**Strategy** allows one of a family of algorithms to be selected on-the-fly at runtime.

**Template** method defines the skeleton of an algorithm as an abstract class, allowing its subclasses to provide concrete behavior.

**Visitor** separates an algorithm from an object structure by moving the hierarchy of methods into one object.

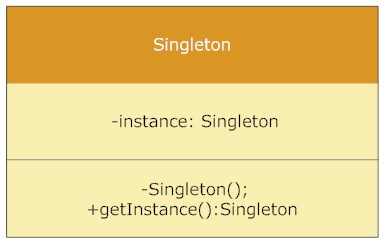
# Singleton Design Pattern – An Introspection w/ Best Practices

Definition:

Singleton is a part of Gang of Four design pattern and it is categorized under creational design patterns.

In this article we are going to take a deeper look into the usage of the Singleton pattern. It is one of the most simple design pattern in terms of the modelling but on the other hand this is one of the most controversial pattern in terms of complexity of usage.

In Java the Singleton pattern will ensure that there is only one instance of a class is created in the Java Virtual Machine. It is used to provide global point of access to the object. In terms of practical use Singleton patterns are used in logging, caches, thread pools, configuration settings, device driver objects. Design pattern is often used in conjunction with Factory design pattern. This pattern is also used in Service Locator JEE pattern.



Singleton Class Diagram

Static member : This contains the instance of the singleton class.

Private constructor : This will prevent anybody else to instantiate the Singleton class.

Static public method : This provides the global point of access to the Singleton object and returns the instance to the client calling class.

Implementation Example: Lazy initialization

Let us look into a singleton implementation example in Java. The below code uses Lazy initialization process.

public class SingletonExample {

// Static member holds only one instance of the

// SingletonExample class

private static SingletonExample singletonInstance;

// SingletonExample prevents any other class from instantiating

private SingletonExample() {

}

// Providing Global point of access

public static SingletonExample getSingletonInstance() {

if (null == singletonInstance) {

singletonInstance = new SingletonExample();

}

return singletonInstance;

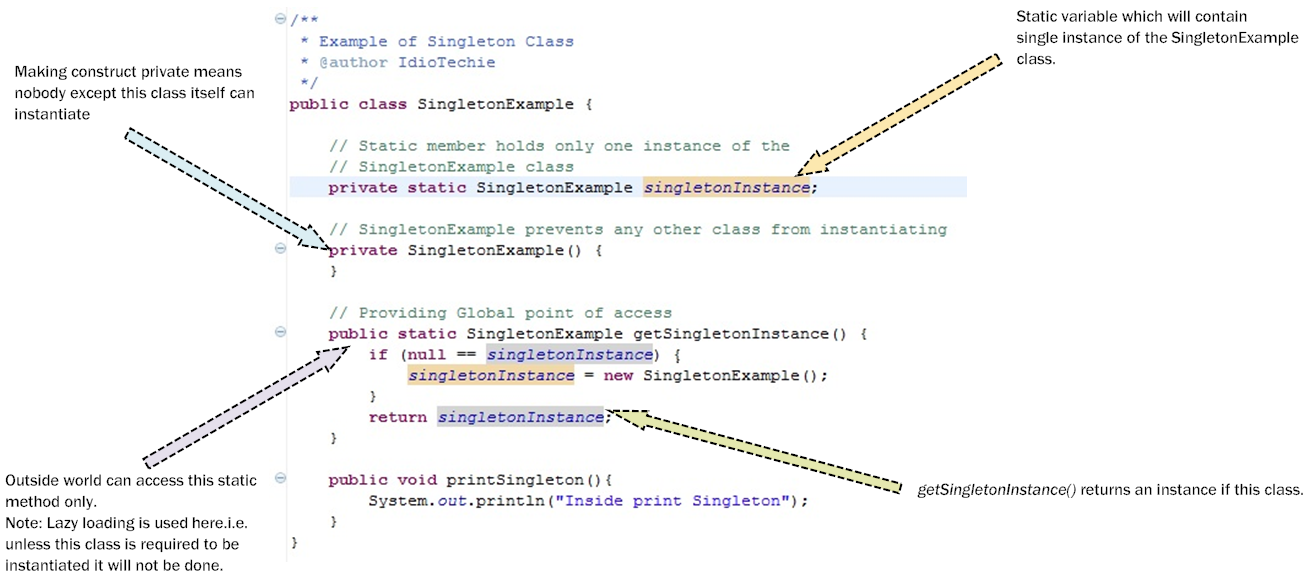
}

public void printSingleton(){

System.out.println("Inside print Singleton");

}

}

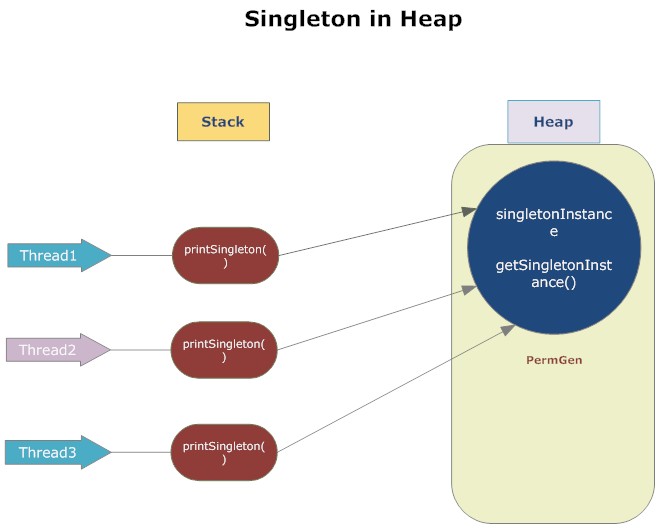


Singleton Pattern Code Explanation

When this class will be called from the client side using SingletonExample.getSingletonInstance().printSingleton(); then at the first time only an instance will be created. During second time onwards for all subsequent calls we will be referring to the same object and the getSingletonInstance() method returns the same instance of the SingletonExample class which was created during the first time. You can test this by adding a print statement the following code as:

Implementation Example: Double check locking

The above code works absolutely fine in a single threaded environment and processes the result faster because of lazy initialization. However the above code might create some abrupt behaviour in the results in a multithreaded environment as in this situation multiple threads can possibly create multiple instance of the same SingletonExample class if they try to access the getSingletonInstance() method at the same time. Imagine a practical scenario where we have to create a log file and update it or while using a shared resource like Printer. To prevent this we must use some locking mechanism so that the second thread cannot use this getInstance() method until the first thread has completed the process.



Singleton in heap

In Figure 3 we show how multiple threads access the singleton instance. Since the singleton instance is a static class variable in the stored in the PermGen space of the heap. This applies to *getSingletonInstance()* instance method as well since it is static too. In the multithreading environment to prevent each thread to create another instance of singleton object and thus creating concurrency issue we will need to use locking mechanism. This can be achieved by synchronized keyword. By using this synchronized keyword we prevent Thread2 or Thread3 to access the singleton instance while Thread1 inside the method*getSingletonInstance()*.

From code perspective it means:

public static synchronized SingletonExample getSingletonInstance() {

if (null == singletonInstance) {

singletonInstance = new SingletonExample();

}

return singletonInstance;

}

So this means that every time the getSingletonInstance() is called it gives us an additional overhead . To prevent this expensive operation we will use double checked locking so that the synchronization happens only during the first call and we limit this expensive operation to happen only once. It is only required for:

singletonInstance = new SingletonExample();

public static volatile SingletonExample getSingletonInstance() {

if (null == singletonInstance) {

synchronized (SingletonExample.class){

if (null == singletonInstance) {

singletonInstance = new SingletonExample();

}

}

}

return singletonInstance;

}

In the above code snippet imagine that multiple threads comes concurrently and tries to create the new instance. In such situation the may be three or more threads are waiting on the synchronized block to get access. Since we have used synchronized only one thread will be given access. All the remaining threads which were waiting on the synchronized block will be given access when first thread exits this block. However when the remaining concurrent thread enters the synchronized block they are prevented to enter further due to the double check : null check. Since the first thread has already created an instance no other thread will enter this loop.

All the remaining threads that were not lucky to enter the synchronized block along with the first thread will be blocked at the first null check. This mechanism is called double checked locking and it provides significant performance benefit and also it is cost effective solution.

Implementation Example: Volatile Keyword

We can also use the volatile keyword to the instance variable declaration.

private volatile static SingletonExample singletonInstance;

The volatile keyword helps as concurrency control tool in a multithreaded environment and provides the latest update in a most accurate manner.However please note that double check locking might not work before Java 5. In such situation we can use early loading mechanism. If we remember about the original sample code we had used lazy loading. In case of early loading we will instantiate the SingletonExample class at the start and this will be referred to the private static instance field.

public class SingletonExample {

private static final SingletonExample singletonInstance = new SingletonExample;

// SingletonExample prevents any other class from instantiating

private SingletonExample() {

}

// Providing Global point of access

public static SingletonExample getSingletonInstance() {

return singletonInstance;

}

public void printSingleton(){

System.out.println("Inside print Singleton");

}

}

In this approach the singleton object is created before it is needed. The JVM takes care of the static variable initialization and ensures that the process is thread safe and that the instance is created before the threads tries to access it. In case of Lazy loading the singletonInstance is created when the client class calls the getSingleInstance() whereas in case of the early loading the singletonInstance is create when the class is loaded in the memory.

Implementation Example: Using enum

Implementing Singleton in Java 5 or above version using Enum:

Enum is thread safe and implementation of Singleton through Enum ensures that your singleton will have only one instance even in a multithreaded environment. Let us see a simple implementation:

public enum SingletonEnum {

02.INSTANCE;

03.public void doStuff(){

04.System.out.println("Singleton using Enum");

05.}

06.}

07.And this can be called from clients :

08.public static void main(String[] args) {

09.SingletonEnum.INSTANCE.doStuff();

10.

11.}

#### **FAQs:**

Question: Why can’t we use a static class instead of singleton?  
Answer:

* One of the key advantages of singleton over static class is that it can implement interfaces and extend classes while the static class cannot (it can extend classes, but it does not inherit their instance members). If we consider a static class it can only be a nested static class as top level class cannot be a static class. Static means that it belongs to a class it is in and not to any instance. So it cannot be a top level class.
* Another difference is that static class will have all its member as static only unlike Singleton.
* Another advantage of Singleton is that it can be lazily loaded whereas static will be initialized whenever it is first loaded.
* Singleton object stores in Heap but, static object stores in stack.
* We can clone the object of Singleton but, we can not clone the static class object.
* Singleton can use the Object Oriented feature of polymorphism but static class cannot.

**Question**: Can the singleton class be subclassed?  
**Answer**: Frankly speaking singleton is just a design pattern and it can be subclassed. However it is worth to understand the logic or requirement behind subclassing a singleton class as the child class might not inherit the singleton pattern objective by extending the Singleton class. However the subclassing can be prevented by using the final keyword in the class declaration.

**Question**: Can there be multiple instance of singleton using cloning?  
**Answer**: That was a good catch! What do we do now? To prevent the another instance to be created of the singleton instance we can throw exception from inside the clone() method.

**Question**: What is the impact if we are creating another instance of singleton using serialization and deserialization?  
**Answer**: When we serialize a class and deserialize it then it creates another instance of the singleton class. Basically as many times as you deserialize the singleton instance it will create multiple instance. Well in this case the best way is to make the singleton as enum. In that way the underlying Java implementation takes care of all the details. If this is not possible then we will need to override the *readobject()* method to return the same singleton instance.

**Question**: Which other pattern works with Singleton?  
**Answer**:There are several other pattern like Factory method, builder and prototype pattern which uses Singleton pattern during the implementation.

**Question**: Which classes in JDK uses singleton pattern?  
Answer: java.lang.Runtime : In every Java application there is only one Runtime instance that allows the application to interface with the environment it is running. The getRuntime is equivalent to the getInstance() method of the singleton class.

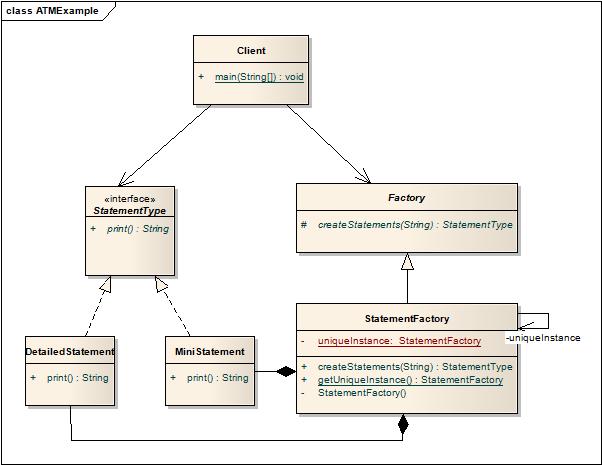
#### **Uses of Singleton design pattern:**

Various usages of Singleton Patterns:

* Hardware interface access: The use of singleton depends on the requirements. However practically singleton can be used in case external hardware resource usage limitation required e.g. Hardware printers where the print spooler can be made a singleton to avoid multiple concurrent accesses and creating deadlock.
* Logger : Similarly singleton is a good potential candidate for using in the log files generation. Imagine an application where the logging utility has to produce one log file based on the messages received from the users. If there is multiple client application using this logging utility class they might create multiple instances of this class and it can potentially cause issues during concurrent access to the same logger file. We can use the logger utility class as a singleton and provide a global point of reference.
* Configuration File: This is another potential candidate for Singleton pattern because this has a performance benefit as it prevents multiple users to repeatedly access and read the configuration file or properties file. It creates a single instance of the configuration file which can be accessed by multiple calls concurrently as it will provide static config data loaded into in-memory objects. The application only reads from the configuration file at the first time and there after from second call onwards the client applications read the data from in-memory objects.
* Cache: We can use the cache as a singleton object as it can have a global point of reference and for all future calls to the cache object the client application will use the in-memory object.

#### **Hands-on:**

Let us take a small practical example to understand the Singleton design pattern in more details.  
  
**Problem Statement:**  
Design a small ATM printing application which can generate multiple types of statements of the transaction including Mini Statement, Detailed statement etc. However the customer should be aware of the creation of these statements. Ensure that the memory consumption is minimized.  
  
**Design Solution:**  
The above requirement can be addressed using two core Gang of four design pattern – Factory design pattern and Singleton design pattern. In order to generate multiple types of statements for the ATM transactions in the ATM machine we can create a Statement Factory object which will have a factory method of *createStatements()*. The createStatements will create DetailedStatement or MiniStatement objects.   
  
The client object will be completely unware of the object creation since it will interact with the Factory interface only. We will also create an interface called StatementType. This will allow further statement type objects e.g. Credit card statement etc to be added. So the solution is scalable and extensible following the object oriented Open/Closed design principle.  
  
The second requirement of reducing the memory consumption can be achieved by using Singleton design pattern. The Statement Factory class need not be initiated multiple times and a single factory can create multiple statement objects. Singleton pattern will create a single instance of the StatementFactory class thus saving memory.



ATM example

* Factory: Factory is an abstract class which is a single point of contact for the client. All the concrete factory classes needs to implement the abstract factory method.
* StatementFactory: This is the Factory implementation class which consist of the creator method. This class extends from the Factory abstract class.This is the main creator class for all the products e.g. Statements.
* StatementType: This is a product interface which provides abstraction to the various types of products which needs to be created by the Factory class.
* DetailedStatement: This is a concrete implementation of the StatementType interface which will print the detailed statements.
* MiniStatement: This is a concrete implementation of the StatementType interface which will print the mini statements.
* Client: This is the client class which will call the StatementFactory and StatementType and request for object creation.

**Assumption:**  
The design solution caters to only single ATM machine.

#### **Sample Code:**

Factory.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)public abstract class Factory {

protected abstract StatementType createStatements(String selection);

}

StatementFactory.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)

public class StatementFactory extends Factory {

private static StatementFactory uniqueInstance;

private StatementFactory() {

}

public static StatementFactory getUniqueInstance() {

if (uniqueInstance == null) {

uniqueInstance = new StatementFactory();

System.out.println("Creating a new StatementFactory instance");

}

return uniqueInstance;

}

public StatementType createStatements(String selection) {

if (selection.equalsIgnoreCase("detailedStmt")) {

return new DetailedStatement();

} else if (selection.equalsIgnoreCase("miniStmt")) {

return new MiniStatement();

}

throw new IllegalArgumentException("Selection doesnot exist");

}

}

StatementType.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)

public interface StatementType {

String print();

}

DetailedStatement.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)

public class DetailedStatement implements StatementType {

@Override

public String print() {

System.out.println("Detailed Statement Created");

return "detailedStmt";

}

}

MiniStatement.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)

public class MiniStatement implements StatementType {

@Override

public String print() {

System.out.println("Mini Statement Created");

return "miniStmt";

}

}

Client.java

[view source](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#viewSource)

[print](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#printSource)[?](http://java.dzone.com/articles/singleton-design-pattern-%E2%80%93?page=0,1#about)

public class Client {

public static void main(String[] args) {

System.out.println("Enter your selection:");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String selection = null;

try {

selection = br.readLine();

} catch (IOException e) {

e.printStackTrace();

}

Factory factory = StatementFactory.getUniqueInstance();

StatementType objStmtType = factory.createStatements(selection);

System.out.println(objStmtType.print());

}

}

**Conclusion:**

In the above articles we have gone into the details of the Singleton pattern, how to implement singleton pattern along with a practical application. Though singleton pattern looks a simple implementation we should resist ourselves from using it until and unless there is a strong requirement for it. You can blame the unpredictable nature of the results in the multi-threading environment. Though we can use enum in Java 5 and above it sometimes get difficult to implement your logic always in enum or there might be legacy code before Java 5. Hope our readers have liked this article.

# Java interview

## **Table of Contents**

* [Object Oriented Programming (OOP)](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#1)
* [General Questions about Java](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#2)
* [Java Threads](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#3)
* [Java Collections](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#4)
* [Garbage Collectors](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#5)
* [Exception Handling](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#6)
* [Java Applets](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#7)
* [Swing](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#8)
* [JDBC](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#9)
* [Remote Method Invocation (RMI)](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#10)
* [Servlets](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#11)
* [JSP](http://www.javacodegeeks.com/2014/04/java-interview-questions-and-answers.html#12)

## **Object Oriented Programming (OOP)**

Java is a computer programming language that is concurrent, class-based and object-oriented. The advantages of object oriented software development are shown below:

* Modular development of code, which leads to easy maintenance and modification.
* Reusability of code.
* Improved reliability and flexibility of code.
* Increased understanding of code.

Object-oriented programming contains many significant features, such as **encapsulation**, **inheritance**, **polymorphism**and [**abstraction**](http://www.javacodegeeks.com/2014/07/abstraction-in-java.html). We analyze each feature separately in the following sections.

#### **Encapsulation**

Encapsulation provides objects with the ability to hide their internal characteristics and behavior. Each object provides a number of methods, which can be accessed by other objects and change its internal data. In Java, there are three access modifiers: public, private and protected. Each modifier imposes different access rights to other classes, either in the same or in external packages. Some of the advantages of using encapsulation are listed below:

* The internal state of every objected is protected by hiding its attributes.
* It increases usability and maintenance of code, because the behavior of an object can be independently changed or extended.
* It improves modularity by preventing objects to interact with each other, in an undesired way.

You can refer to our tutorial [here](http://examples.javacodegeeks.com/java-basics/encapsulation-in-java/) for more details and examples on encapsulation.

#### **Polymorphism**

Polymorphism is the ability of programming languages to present the same interface for differing underlying data types. A polymorphic type is a type whose operations can also be applied to values of some other type.

#### **Inheritance**

Inheritance provides an object with the ability to acquire the fields and methods of another class, called base class. Inheritance provides re-usability of code and can be used to add additional features to an existing class, without modifying it.

#### **Abstraction**

[Abstraction](http://www.javacodegeeks.com/2014/04/why-abstraction-is-really-important.html) is the process of separating ideas from specific instances and thus, develop classes in terms of their own functionality, instead of their implementation details. Java supports the creation and existence of abstract classes that expose interfaces, without including the actual implementation of all methods. The abstraction technique aims to separate the implementation details of a class from its behavior.

#### **Differences between Abstraction and Encapsulation**

Abstraction and encapsulation are complementary concepts. On the one hand, abstraction focuses on the behavior of an object. On the other hand, encapsulation focuses on the implementation of an object’s behavior. Encapsulation is usually achieved by hiding information about the internal state of an object and thus, can be seen as a strategy used in order to provide abstraction.

## **General Questions about Java**

**1. What is JVM ? Why is Java called the “Platform Independent Programming Language” ?** A Java virtual machine (JVM) is a process [virtual machine](http://www.javacodegeeks.com/2013/12/part-1-of-3-synopsis-of-articles-videos-on-performance-tuning-jvm-gc-in-java-mechanical-sympathy-et-al.html) that can execute Java [bytecode](http://www.javacodegeeks.com/2013/12/mastering-java-bytecode.html). Each Java source file is compiled into a bytecode file, which is executed by the JVM. Java was designed to allow application programs to be built that could be run on any platform, without having to be rewritten or recompiled by the programmer for each separate platform. A Java virtual machine makes this possible, because it is aware of the specific instruction lengths and other particularities of the underlying hardware platform.

**2. What is the Difference between JDK and JRE ?** The Java Runtime Environment (JRE) is basically the Java Virtual Machine (JVM) where your Java programs are being executed. It also includes browser plugins for applet execution. The Java Development Kit (JDK) is the full featured Software Development Kit for Java, including the JRE, the compilers and tools (like [JavaDoc](http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html), and [Java Debugger](http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jdb.html)), in order for a user to develop, compile and execute Java applications.

**3. What does the “static” keyword mean ? Can you override private or static method in Java ?** The static keyword denotes that a member variable or method can be accessed, without requiring an instantiation of the class to which it belongs. A user cannot override [static methods in Java](http://www.javacodegeeks.com/2012/05/java-static-methods-can-be-code-smell.html), because method overriding is based upon dynamic binding at runtime and static methods are statically binded at compile time. A static method is not associated with any instance of a class so the concept is not applicable.

**4. Can you access non static variable in static context ?** A static variable in Java belongs to its class and its value remains the same for all its instances. A static variable is initialized when the class is loaded by the JVM. If your code tries to access a non-static variable, without any instance, the compiler will complain, because those variables are not created yet and they are not associated with any instance.

**5. What are the Data Types supported by Java ? What is Autoboxing and Unboxing ?** The eight primitive data types supported by the Java programming language are:

* byte
* short
* int
* long
* float
* double
* boolean
* char

Autoboxing is the [automatic conversion made by the Java compiler](http://www.javacodegeeks.com/2013/07/java-generics-tutorial-example-class-interface-methods-wildcards-and-much-more.html) between the primitive types and their corresponding object wrapper classes. For example, the compiler converts an int to an [Integer](http://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html?is-external=true), a double to a [Double](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html), and so on. If the conversion goes the other way, this operation is called unboxing.

**6. What is Function Overriding and Overloading in Java ?** Method overloading in Java occurs when two or more methods in the same class have the exact same name, but different parameters. On the other hand, method overriding is defined as the case when a child class redefines the same method as a parent class. Overridden methods must have the same name, argument list, and return type. The overriding method may not limit the access of the method it overrides.

**7. What is a Constructor, Constructor Overloading in Java and Copy-Constructor ?** A constructor gets invoked when a new object is created. Every class [has a constructor](http://www.javacodegeeks.com/2014/01/which-is-better-option-cloning-or-copy-constructors.html). In case the programmer does not provide a constructor for a class, the Java compiler (Javac) creates a default constructor for that class. The constructor overloading is similar to method overloading in Java. Different constructors can be created for a single class. Each constructor must have its own unique parameter list. Finally, Java does support copy constructors like C++, but the difference lies in the fact that Java doesn’t create a default copy constructor if you don’t write your own.

**8. Does Java support multiple inheritance ?** No, Java does not support multiple inheritance. Each class is able to extend only on one class, but is able to implement more than one interfaces.

**9. What is the difference between an Interface and an Abstract class ?** Java provides and supports the creation both of [abstract classes](http://examples.javacodegeeks.com/java-basics/java-abstract-class-example/) and interfaces. Both implementations share some common characteristics, but they differ in the following features:

* All methods in an interface are implicitly abstract. On the other hand, an abstract class may contain both abstract and non-abstract methods.
* A class may implement a number of Interfaces, but can extend only one abstract class.
* In order for a class to implement an interface, it must implement all its declared methods. However, a class may not implement all declared methods of an abstract class. Though, in this case, the sub-class must also be declared as abstract.
* Abstract classes can implement interfaces without even providing the implementation of interface methods.
* Variables declared in a Java interface is by default final. An abstract class may contain non-final variables.
* Members of a Java interface are public by default. A member of an abstract class can either be private, protected or public.
* An interface is absolutely abstract and cannot be instantiated. An abstract class also cannot be instantiated, but can be invoked if it contains a main method.

Also check out the [Abstract class and Interface differences for JDK 8](http://www.javacodegeeks.com/2014/04/abstract-class-versus-interface-in-the-jdk-8-era.html).

**10. What are pass by reference and pass by value ?** When an object is passed by value, this means that a copy of the object is passed. Thus, even if changes are made to that object, it doesn’t affect the original value. When an object is passed by reference, this means that the actual object is not passed, rather a reference of the object is passed. Thus, any changes made by the external method, are also reflected in all places.

## **Java Threads**

**11. What is the difference between processes and threads ?** A process is an execution of a program, while a[Thread](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) is a single execution sequence within a process. A process can contain multiple threads. A [Thread](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) is sometimes called a lightweight process.

**12. Explain different ways of creating a thread. Which one would you prefer and why ?** There are three ways that can be used in order for a [Thread](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) to be created:

* A class may extend the [Thread](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) class.
* A class may implement the [Runnable](http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html) interface.
* An application can use the [Executor](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executor.html) framework, in order to create a thread pool.

The [Runnable](http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html) interface is preferred, as it does not require an object to inherit the [Thread](http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html) class. In case your application design requires multiple inheritance, only interfaces can help you. Also, the thread pool is very efficient and can be implemented and used very easily.

**13. Explain the available thread states in a high-level.** During its execution, a thread can reside in one of the following [states](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html):

* [**NEW**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#NEW): The thread becomes ready to run, but does not necessarily start running immediately.
* [**RUNNABLE**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#RUNNABLE): The Java Virtual Machine (JVM) is actively executing the thread’s code.
* [**BLOCKED**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#BLOCKED): The thread is in a blocked state while waiting for a monitor lock.
* [**WAITING**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#WAITING): The thread waits for another thread to perform a particular action.
* [**TIMED\_WAITING**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#TIMED_WAITING): The thread waits for another thread to perform a particular action up to a specified waiting time.
* [**TERMINATED**](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html#TERMINATED): The thread has finished its execution.

**14. What is the difference between a synchronized method and a synchronized block ?** In Java programming, each object has a lock. A thread can acquire the lock for an object by using the synchronized keyword. The synchronized keyword can be applied in a method level (coarse grained lock) or block level of code (fine grained lock).

**15. How does thread synchronization occurs inside a monitor ? What levels of synchronization can you apply ?** The JVM uses locks in conjunction with monitors. A monitor is basically a guardian that watches over a sequence of synchronized code and ensuring that only one thread at a time executes a synchronized piece of code. Each monitor is associated with an object reference. The thread is not allowed to execute the code until it obtains the lock.

**16. What’s a deadlock ?** A condition that occurs when [two processes are waiting for each other to complete](http://www.javacodegeeks.com/2013/01/java-deadlock-example-how-to-analyze-deadlock-situation.html), before proceeding. The result is that both processes wait endlessly.

**17. How do you ensure that N threads can access N resources without deadlock ?** A very simple way to avoid deadlock while using N threads is to impose an ordering on the locks and force each thread to follow that ordering. Thus, if all threads lock and unlock the mutexes in the same order, no deadlocks can arise.

## **Java Collections**

**18. What are the basic interfaces of Java Collections Framework ?** [Java Collections Framework](http://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html" \o ") provides a well designed set of interfaces and classes that support operations on a collections of objects. The most basic interfaces that reside in the Java Collections Framework are:

* [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html), which represents a group of objects known as its elements.
* [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html), which is a collection that cannot contain duplicate elements.
* [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html), which is an ordered collection and can contain duplicate elements.
* [Map](http://docs.oracle.com/javase/7/docs/api/java/util/Map.html), which is an object that maps keys to values and cannot contain duplicate keys.

**19. Why Collection doesn’t extend Cloneable and Serializable interfaces ?** The [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) interface specifies groups of objects known as elements. Each concrete implementation of a [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) can choose its own way of how to maintain and order its elements. Some collections allow duplicate keys, while some other collections don’t. The semantics and the implications of either cloning or serialization come into play when dealing with actual implementations. Thus, the concrete implementations of collections should decide how they can be cloned or serialized.

**20. What is an Iterator ?** The [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) interface provides a number of methods that are able to iterate over any[Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html). Each Java [Collection](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html) contains the [iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html#iterator%28%29) method that returns an [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) instance. Iterators are[capable of removing elements from the underlying collection](http://www.javacodegeeks.com/2011/05/avoid-concurrentmodificationexception.html) during the iteration. **21. What differences exist between Iterator and ListIterator ?** The differences of these elements are listed below:

* An [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) can be used to traverse the [Set](http://docs.oracle.com/javase/7/docs/api/java/util/Set.html) and [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html) collections, while the [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) can be used to iterate only over [Lists](http://docs.oracle.com/javase/7/docs/api/java/util/List.html).
* The [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) can traverse a collection only in forward direction, while the [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) can traverse a [List](http://docs.oracle.com/javase/7/docs/api/java/util/List.html) in both directions.
* The [ListIterator](http://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html) implements the [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) interface and contains extra functionality, such as adding an element, replacing an element, getting the index position for previous and next elements, etc.

**22. What is difference between fail-fast and fail-safe ?** The [Iterator's](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) fail-safe property works with the clone of the underlying collection and thus, it is not affected by any modification in the collection. All the collection classes in java.util package are fail-fast, while the collection classes in java.util.concurrent are fail-safe. Fail-fast iterators throw a[ConcurrentModificationException](http://examples.javacodegeeks.com/java-basics/exceptions/java-util-concurrentmodificationexception-how-to-handle-concurrent-modification-exception/), while fail-safe iterator never throws such an exception.

**23. How HashMap works in Java ?** A [HashMap in Java stores key-value pairs](http://www.javacodegeeks.com/2014/03/how-hashmap-works-in-java.html). The [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) requires a hash function and uses [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29) and equals methods, in order to put and retrieve elements to and from the collection respectively. When the put method is invoked, the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) calculates the hash value of the key and stores the pair in the appropriate index inside the collection. If the key exists, its value is updated with the new value. Some important characteristics of a[HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) are its capacity, its load factor and the threshold resizing.

**24. What is the importance of hashCode() and equals() methods ?** In Java, a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) uses the [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29) and[equals](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals%28java.lang.Object%29) methods to determine the index of the key-value pair and to detect duplicates. More specifically, the [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29)method is used in order to determine where the specified key will be stored. Since different keys may produce the same hash value, the [equals](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals%28java.lang.Object%29) method is used, in order to determine whether the specified key actually exists in the collection or not. Therefore, the implementation of both methods is crucial to the accuracy and efficiency of the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html).

**25. What differences exist between HashMap and Hashtable ?** Both the [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) and [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) classes implement the Map interface and thus, have very similar characteristics. However, they differ in the following features:

* A [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) allows the existence of null keys and values, while a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) doesn’t allow neither null keys, nor null values.
* A [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) is synchronized, while a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is not. Thus, [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is preferred in single-threaded environments, while a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) is suitable for multi-threaded environments.
* A [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) provides its set of keys and a Java application can iterate over them. Thus, a [HashMap](http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html) is fail-fast. On the other hand, a [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) provides an [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html) of its keys.
* The [Hashtable](http://docs.oracle.com/javase/7/docs/api/java/util/Hashtable.html) class is considered to be a legacy class.

**26. What is difference between Array and ArrayList ? When will you use Array over ArrayList ?** The [Array](http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html)and [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) classes differ on the following features:

* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) can contain primitive or objects, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) can contain only objects.
* [Arrays](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html) have fixed size, while an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) is dynamic.
* An [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)provides more methods and features, such as [addAll](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#addAll(java.util.Collection)), [removeAll](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#removeAll(java.util.Collection)), [iterator](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#iterator()), etc.
* For a list of primitive data types, the collections use autoboxing to reduce the coding effort. However, this approach makes them slower when working on fixed size primitive data types.

**27. What is difference between ArrayList and LinkedList ?** Both the [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) and [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) classes implement the List interface, but they differ on the following features:

* An [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html) is an index based data structure backed by an [Array](http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html). It provides random access to its elements with a performance equal to O(1). On the other hand, a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) stores its data as list of elements and every element is linked to its previous and next element. In this case, the search operation for an element has execution time equal to O(n).
* The Insertion, addition and removal operations of an element are faster in a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) compared to an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), because there is no need of resizing an array or updating the index when an element is added in some arbitrary position inside the collection.
* A [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) consumes more memory than an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html), because every node in a [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) stores two references, one for its previous element and one for its next element.

Check also our article [ArrayList vs. LinkedList](http://www.javacodegeeks.com/2013/12/arraylist-vs-linkedlist.html).

**28. What is Comparable and Comparator interface ? List their differences.** Java provides the [Comparable](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html)interface, which contains only one method, called [compareTo](http://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html#compareTo(T)). This method compares two objects, in order to impose an order between them. Specifically, it returns a negative integer, zero, or a positive integer to indicate that the input object is less than, equal or greater than the existing object. Java provides the [Comparator](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html) interface, which contains two methods, called [compare](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html#compare(T,%20T)) and [equals](http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html#equals(java.lang.Object)). The first method compares its two input arguments and imposes an order between them. It returns a negative integer, zero, or a positive integer to indicate that the first argument is less than, equal to, or greater than the second. The second method requires an object as a parameter and aims to decide whether the input object is equal to the comparator. The method returns true, only if the specified object is also a comparator and it imposes the same ordering as the comparator.

**29. What is Java Priority Queue ?** The [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) is an unbounded queue, based on a priority heap and its elements are ordered in their natural order. At the time of its creation, we can provide a Comparator that is responsible for ordering the elements of the [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html). A [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) doesn’t allow [null values](http://examples.javacodegeeks.com/java-basics/exceptions/java-lang-nullpointerexception-how-to-handle-null-pointer-exception/), those objects that doesn’t provide natural ordering, or those objects that don’t have any comparator associated with them. Finally, the Java [PriorityQueue](http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html) is not thread-safe and it requires O(log(n)) time for its enqueing and dequeing operations.

**30. What do you know about the big-O notation and can you give some examples with respect to different data structures ?** The [Big-O notation](http://www.javacodegeeks.com/2011/04/simple-big-o-notation-post.html) simply describes how well an algorithm scales or performs in the worst case scenario as the number of elements in a data structure increases. The Big-O notation can also be used to describe other behavior such as memory consumption. Since the collection classes are actually data structures, we usually use the Big-O notation to chose the best implementation to use, based on time, memory and performance. Big-O notation can give a good indication about performance for large amounts of data.

**31. What is the tradeoff between using an unordered array versus an ordered array ?** The major advantage of an ordered array is that the search times have time complexity of O(log n), compared to that of an unordered array, which is O (n). The disadvantage of an ordered array is that the insertion operation has a time complexity of O(n), because the elements with higher values must be moved to make room for the new element. Instead, the insertion operation for an unordered array takes constant time of O(1).

**32. What are some of the best practices relating to the Java Collection framework ?**

* Choosing the right type of the collection to use, based on the application’s needs, is very crucial for its performance. For example if the size of the elements is fixed and know a priori, we shall use an [Array](http://docs.oracle.com/javase/7/docs/api/java/lang/reflect/Array.html), instead of an [ArrayList](http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html).
* Some collection classes allow us to specify their initial capacity. Thus, if we have an estimation on the number of elements that will be stored, we can use it to avoid rehashing or resizing.
* Always use Generics for type-safety, readability, and robustness. Also, by using Generics you avoid the[ClassCastException](http://docs.oracle.com/javase/7/docs/api/java/lang/ClassCastException.html) during runtime.
* Use immutable classes provided by the Java Development Kit (JDK) as a key in a Map, in order to avoid the implementation of the [hashCode](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29) and equals methods for our custom class.
* Program in terms of interface not implementation.
* Return zero-length collections or arrays as opposed to returning a null in case the underlying collection is actually empty.

**33. What’s the difference between Enumeration and Iterator interfaces ?** [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html" \o ") is twice as fast as compared to an Iterator and uses very less memory. However, the [Iterator](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html) is much safer compared to [Enumeration](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html), because other threads are not able to modify the collection object that is currently traversed by the iterator. Also,[Iterators](http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html" \o ")allow the caller to remove elements from the underlying collection, something which is not possible with[Enumerations](http://docs.oracle.com/javase/7/docs/api/java/util/Enumeration.html).

**34. What is the difference between HashSet and TreeSet ?** The [HashSet](http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) is Implemented using a hash table and thus, its elements are not ordered. The add, remove, and contains methods of a [HashSet](http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) have constant time complexity O(1). On the other hand, a [TreeSet](http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) is implemented using a tree structure. The elements in a [TreeSet](http://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html) are sorted, and thus, the add, remove, and contains methods have time complexity of O(logn).

## **Garbage Collectors**

**35. What is the purpose of garbage collection in Java, and when is it used ?** The purpose of garbage collection is to identify and discard those objects that are no longer needed by the application, in order for the resources to be reclaimed and reused.

**36. What does System.gc() and Runtime.gc() methods do ?** These methods can be used as a hint to the JVM, in order to start a garbage collection. However, this it is up to the Java Virtual Machine (JVM) to start the garbage collection immediately or later in time.

**37. When is the finalize() called ? What is the purpose of finalization ?** The finalize method is called by the garbage collector, just before releasing the object’s memory. It is normally advised to release resources held by the object inside the finalize method.

**38. If an object reference is set to null, will the Garbage Collector immediately free the memory held by that object ?** No, the object will be available for garbage collection in the next cycle of the garbage collector.

**39. What is structure of Java Heap ? What is Perm Gen space in Heap ?** The [JVM has a heap](http://www.javacodegeeks.com/2012/07/5-tips-for-proper-java-heap-size.html) that is the runtime data area from which memory for all class instances and arrays is allocated. It is created at the JVM start-up. Heap memory for objects is reclaimed by an automatic memory management system which is known as a garbage collector. Heap memory consists of live and dead objects. Live objects are accessible by the application and will not be a subject of garbage collection. Dead objects are those which will never be accessible by the application, but have not been collected by the garbage collector yet. Such objects occupy the heap memory space until they are eventually collected by the garbage collector.

**40. What is the difference between Serial and Throughput Garbage collector ?** The throughput garbage collector uses a parallel version of the young generation collector and is meant to be used with applications that have medium to large data sets. On the other hand, the serial collector is usually adequate for most small applications (those requiring heaps of up to approximately 100MB on modern processors).

**41. When does an Object becomes eligible for Garbage collection in Java ?** A Java object is subject to garbage collection when it becomes unreachable to the program in which it is currently used.

**42. Does Garbage collection occur in permanent generation space in JVM ?** Garbage Collection does occur in PermGen space and if PermGen space is full or cross a threshold, it can trigger a full garbage collection. If you look carefully at the output of the garbage collector, you will find that PermGen space is also garbage collected. This is the reason why correct sizing of PermGen space is important to avoid frequent full garbage collections. Also check our article [Java 8: PermGen to Metaspace](http://www.javacodegeeks.com/2013/02/java-8-from-permgen-to-metaspace.html).

## **Exception Handling**

**43. What are the two types of Exceptions in Java ? Which are the differences between them ?** Java has two types of exceptions: checked exceptions and unchecked exceptions. Unchecked exceptions do not need to be declared in a method or a constructor’s throws clause, if they can be thrown by the execution of the method or the constructor, and propagate outside the method or constructor boundary. On the other hand, checked exceptions must be declared in a method or a constructor’s throws clause. See here for tips on [Java exception handling](http://www.javacodegeeks.com/2013/07/java-exception-handling-tutorial-with-examples-and-best-practices.html).

**44. What is the difference between Exception and Error in java ?** [Exception](http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html" \o ") and [Error](http://docs.oracle.com/javase/7/docs/api/java/lang/Error.html) classes are both subclasses of the [Throwable](http://docs.oracle.com/javase/7/docs/api/java/lang/Throwable.html) class. The [Exception](http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html) class is used for exceptional conditions that a user’s program should catch. The [Error](http://docs.oracle.com/javase/7/docs/api/java/lang/Error.html) class defines exceptions that are not excepted to be caught by the user program.

**45. What is the difference between throw and throws ?** The throw keyword is used to explicitly raise a exception within the program. On the contrary, the throws clause is used to indicate those exceptions that are not handled by a method. Each method must explicitly specify which exceptions does not handle, so the callers of that method can guard against possible exceptions. Finally, multiple exceptions are separated by a comma.

**45. What is the importance of finally block in exception handling ?** A finally block will always be executed, whether or not an exception is actually thrown. Even in the case where the catch statement is missing and an exception is thrown, the finally block will still be executed. Last thing to mention is that the finally block is used to release resources like I/O buffers, database connections, etc.

**46. What will happen to the Exception object after exception handling ?** The [Exception](http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html) object will be garbage collected in the next garbage collection.

**47. How does finally block differ from finalize() method ?** A finally block will be executed whether or not an exception is thrown and is used to release those resources held by the application. Finalize is a protected method of the Object class, which is called by the Java Virtual Machine (JVM) just before an object is garbage collected.

## **Java Applets**

**48. What is an Applet ?** A java applet is program that can be included in a HTML page and be executed in a java enabled client browser. Applets are used for creating dynamic and interactive web applications.

**49. Explain the life cycle of an Applet.** An applet may undergo the following states:

* **Init**: An applet is initialized each time is loaded.
* **Start**: Begin the execution of an applet.
* **Stop**: Stop the execution of an applet.
* **Destroy**: Perform a final cleanup, before unloading the applet.

**50. What happens when an applet is loaded ?** First of all, an instance of the applet’s controlling class is created. Then, the applet initializes itself and finally, it starts running.

**51. What is the difference between an Applet and a Java Application ?** Applets are executed within a java enabled browser, but a Java application is a standalone Java program that can be executed outside of a browser. However, they both require the existence of a Java Virtual Machine (JVM). Furthermore, a Java application requires a main method with a specific signature, in order to start its execution. Java applets don’t need such a method to start their execution. Finally, Java applets typically use a restrictive security policy, while Java applications usually use more relaxed security policies.

**52. What are the restrictions imposed on Java applets ?** Mostly due to security reasons, the following restrictions are imposed on Java applets:

* An applet cannot load libraries or define native methods.
* An applet cannot ordinarily read or write files on the execution host.
* An applet cannot read certain system properties.
* An applet cannot make network connections except to the host that it came from.
* An applet cannot start any program on the host that’s executing it.

**53. What are untrusted applets ?** Untrusted applets are those Java applets that cannot access or execute local system files. By default, all downloaded applets are considered as untrusted.

**54. What is the difference between applets loaded over the internet and applets loaded via the file system ?**Regarding the case where an applet is loaded over the internet, the applet is loaded by the applet classloader and is subject to the restrictions enforced by the applet security manager. Regarding the case where an applet is loaded from the client’s local disk, the applet is loaded by the file system loader. Applets loaded via the file system are allowed to read files, write files and to load libraries on the client. Also, applets loaded via the file system are allowed to execute processes and finally, applets loaded via the file system are not passed through the byte code verifier.

**55. What is the applet class loader, and what does it provide ?** When an applet is loaded over the internet, the applet is loaded by the applet classloader. The class loader enforces the Java name space hierarchy. Also, the class loader guarantees that a unique namespace exists for classes that come from the local file system, and that a unique namespace exists for each network source. When a browser loads an applet over the net, that applet’s classes are placed in a private namespace associated with the applet’s origin. Then, those classes loaded by the class loader are passed through the verifier.The verifier checks that the class file conforms to the Java language specification . Among other things, the verifier ensures that there are no stack overflows or underflows and that the parameters to all bytecode instructions are correct.

**56. What is the applet security manager, and what does it provide ?** The applet security manager is a mechanism to impose restrictions on Java applets. A browser may only have one security manager. The security manager is established at startup, and it cannot thereafter be replaced, overloaded, overridden, or extended.

## [**Swing**](http://examples.javacodegeeks.com/desktop-java/swing/)

**57. What is the difference between a Choice and a List ?** A Choice is displayed in a compact form that must be pulled down, in order for a user to be able to see the list of all available choices. Only one item may be selected from a Choice. A [List](http://examples.javacodegeeks.com/desktop-java/swing/jlist/create-jlist-example/) may be displayed in such a way that several List items are visible. A List supports the selection of one or more List items.

**58. What is a layout manager ?** A layout manager is the used to organize the components in a container.

**59. What is the difference between a Scrollbar and a JScrollPane ?** A [Scrollbar](http://docs.oracle.com/javase/7/docs/api/java/awt/Scrollbar.html) is a [Component](http://docs.oracle.com/javase/7/docs/api/java/awt/Component.html), but not a[Container](http://docs.oracle.com/javase/7/docs/api/java/awt/Container.html). A [ScrollPane](http://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollPane.html) is a [Container](http://docs.oracle.com/javase/7/docs/api/java/awt/Container.html). A [ScrollPane](http://docs.oracle.com/javase/7/docs/api/javax/swing/JScrollPane.html) handles its own events and performs its own scrolling.

**60. Which Swing methods are thread-safe ?** There are only three thread-safe methods: repaint, revalidate, and invalidate.

**61. Name three Component subclasses that support painting.** The [Canvas](http://docs.oracle.com/javase/7/docs/api/java/awt/Canvas.html), [Frame](http://docs.oracle.com/javase/7/docs/api/java/awt/Frame.html), [Panel](http://docs.oracle.com/javase/7/docs/api/java/awt/Panel.html), and Applet classes support painting.

**62. What is clipping ?** Clipping is defined as the process of confining paint operations to a limited area or shape.

**63. What is the difference between a MenuItem and a CheckboxMenuItem ?** The [CheckboxMenuItem](http://docs.oracle.com/javase/7/docs/api/java/awt/CheckboxMenuItem.html) class extends the [MenuItem](http://docs.oracle.com/javase/7/docs/api/java/awt/MenuItem.html) class and supports a menu item that may be either checked or unchecked.

**64. How are the elements of a BorderLayout organized ?** The elements of a [BorderLayout](http://docs.oracle.com/javase/7/docs/api/java/awt/BorderLayout.html) are organized at the borders (North, South, East, and West) and the center of a container.

**65. How are the elements of a GridBagLayout organized ?** The elements of a [GridBagLayout](http://docs.oracle.com/javase/7/docs/api/java/awt/GridBagLayout.html) are organized according to a grid. The elements are of different sizes and may occupy more than one row or column of the grid. Thus, the rows and columns may have different sizes.

**66. What is the difference between a Window and a Frame ?** The [Frame](http://docs.oracle.com/javase/7/docs/api/java/awt/Frame.html) class extends the [Window](http://docs.oracle.com/javase/7/docs/api/java/awt/Window.html) class and defines a main application window that can have a menu bar.

**67. What is the relationship between clipping and repainting ?** When a window is repainted by the AWT painting thread, it sets the clipping regions to the area of the window that requires repainting.

**68. What is the relationship between an event-listener interface and an event-adapter class ?** An event-listener interface defines the methods that must be implemented by an event handler for a particular event. An event adapter provides a default implementation of an event-listener interface.

**69. How can a GUI component handle its own events ?** A GUI component can handle its own events, by implementing the corresponding event-listener interface and adding itself as its own event listener.

**70. What advantage do Java’s layout managers provide over traditional windowing systems ?** Java uses layout managers to lay out components in a consistent manner, across all windowing platforms. Since layout managers aren’t tied to absolute sizing and positioning, they are able to accomodate platform-specific differences among windowing systems.

**71. What is the design pattern that Java uses for all Swing components ?** The design pattern used by Java for all Swing components is the Model View Controller (MVC) pattern.

## **JDBC**

**72. What is JDBC ?** JDBC is an abstraction layer that allows users to choose between databases. [JDBC enables developers to write database applications in Java](http://www.javacodegeeks.com/2014/03/java-8-friday-java-8-will-revolutionize-database-access.html), without having to concern themselves with the underlying details of a particular database.

**73. Explain the role of Driver in JDBC.** The JDBC Driver provides vendor-specific implementations of the abstract classes provided by the JDBC API. Each driver must provide implementations for the following classes of the java.sql package:[Connection](http://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html), [Statement](http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html), [PreparedStatement](http://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html), [CallableStatement](http://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html), [ResultSet](http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html) and [Driver](http://docs.oracle.com/javase/7/docs/api/java/sql/Driver.html).

**74. What is the purpose Class.forName method ?** This method is used to method is used to load the driver that will establish a connection to the database.

**75. What is the advantage of PreparedStatement over Statement ?** PreparedStatements are precompiled and thus,[their performance is much better](http://examples.javacodegeeks.com/core-java/sql/batch-statement-execution-example/). Also, PreparedStatement objects can be reused with different input values to their queries.

**76. What is the use of CallableStatement ? Name the method, which is used to prepare a CallableStatement.**A [CallableStatement](http://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html) is used to execute stored procedures. Stored procedures are stored and offered by a database. Stored procedures may take input values from the user and may return a result. The usage of stored procedures is highly encouraged, because it offers security and modularity.The method that prepares a [CallableStatement](http://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html) is the following:

|  |  |
| --- | --- |
| 1 | CallableStament.prepareCall(); |

**77. What does Connection pooling mean ?** The interaction with a database can be costly, regarding the opening and closing of database connections. Especially, when the number of database clients increases, this cost is very high and a large number of resources is consumed.A pool of database connections is obtained at start up by the application server and is maintained in a pool. A request for a connection is served by a [connection residing in the pool](http://examples.javacodegeeks.com/enterprise-java/hibernate/hibernate-connection-pool-configuration-with-c3p0-example/). In the end of the connection, the request is returned to the pool and can be used to satisfy future requests.

## **Remote Method Invocation (RMI)**

**78. What is RMI ?** The Java Remote Method Invocation (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection. Remote Method Invocation (RMI) can also be seen as the process of activating a method on a remotely running object. RMI offers location transparency because a user feels that a method is executed on a locally running object. Check some [RMI Tips here](http://www.javacodegeeks.com/2013/11/two-things-to-remember-when-using-java-rmi.html).

**79. What is the basic principle of RMI architecture ?** The RMI architecture is based on a very important principle which states that the definition of the behavior and the implementation of that behavior, are separate concepts. RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.

**80. What are the layers of RMI Architecture ?** The RMI architecture consists of the following layers:

* **Stub and Skeleton layer**: This layer lies just beneath the view of the developer. This layer is responsible for intercepting method calls made by the client to the interface and redirect these calls to a remote RMI Service.
* **Remote Reference Layer**: The second layer of the RMI architecture deals with the interpretation of references made from the client to the server’s remote objects. This layer interprets and manages references made from clients to the remote service objects. The connection is a one-to-one (unicast) link.
* **Transport layer**: This layer is responsible for connecting the two JVM participating in the service. This layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

**81. What is the role of Remote Interface in RMI ?** The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. A class that implements a remote interface should declare the remote interfaces being implemented, define the constructor for each remote object and provide an implementation for each remote method in all remote interfaces.

**82. What is the role of the java.rmi.Naming Class ?** The java.rmi.Naming class provides methods for storing and obtaining references to remote objects in the remote object registry. Each method of the Naming class takes as one of its arguments a name that is a String in URL format.

**83. What is meant by binding in RMI ?** Binding is the process of associating or registering a name for a remote object, which can be used at a later time, in order to look up that remote object. A remote object can be associated with a name using the bind or rebind methods of the Naming class.

**84. What is the difference between using bind() and rebind() methods of Naming Class ?** The bind method bind is responsible for binding the specified name to a remote object, while the rebind method is responsible for rebinding the specified name to a new remote object. In case a binding exists for that name, the binding is replaced.

**85. What are the steps involved to make work a RMI program ?** The following steps must be involved in order for a RMI program to work properly:

* Compilation of all source files.
* Generatation of the stubs using rmic.
* Start the rmiregistry.
* Start the RMIServer.
* Run the client program.

**86. What is the role of stub in RMI ?** A stub for a remote object acts as a client’s local representative or proxy for the remote object. The caller invokes a method on the local stub, which is responsible for executing the method on the remote object. When a stub’s method is invoked, it undergoes the following steps:

* It initiates a connection to the remote JVM containing the remote object.
* It marshals the parameters to the remote JVM.
* It waits for the result of the method invocation and execution.
* It unmarshals the return value or an exception if the method has not been successfully executed.
* It returns the value to the caller.

**87. What is DGC ? And how does it work ?** DGC stands for Distributed Garbage Collection. Remote Method Invocation (RMI) uses DGC for automatic garbage collection. Since RMI involves remote object references across JVM’s, garbage collection can be quite difficult. DGC uses a reference counting algorithm to provide automatic memory management for remote objects.

**88. What is the purpose of using RMISecurityManager in RMI ?** RMISecurityManager provides a security manager that can be used by RMI applications, which use downloaded code. The class loader of RMI will not download any classes from remote locations, if the security manager has not been set.

**89. Explain Marshalling and demarshalling.** When an application wants to pass its memory objects across a network to another host or persist it to storage, the in-memory representation must be converted to a suitable format. This process is called marshalling and the revert operation is called demarshalling.

**90. Explain Serialization and Deserialization.** Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes and includes the object’s data, as well as information about the object’s type, and the types of data stored in the object. Thus, serialization can be seen as a way of flattening objects, in order to be stored on disk, and later, read back and reconstituted. Deserialisation is the reverse process of converting an object from its flattened state to a live object.

## **Servlets**

**91. What is a Servlet ?** [The servlet](http://examples.javacodegeeks.com/enterprise-java/servlet/sample-java-servlet/) is a Java programming language class used to process client requests and generate dynamic web content. Servlets are mostly used to process or store data submitted by an HTML form, provide dynamic content and manage state information that does not exist in the stateless HTTP protocol.

**92. Explain the architechure of a Servlet.** The core abstraction that must be implemented by all servlets is the javax.servlet.Servlet interface. Each servlet must implement it either directly or indirectly, either by extending javax.servlet.GenericServlet or javax.servlet.http.HTTPServlet. Finally, each servlet is able to serve multiple requests in parallel using multithreading.

**93. What is the difference between an Applet and a Servlet ?** An Applet is a client side java program that runs within a Web browser on the client machine. On the other hand, a servlet is a server side component that runs on the web server.An applet can use the user interface classes, while a servlet does not have a user interface. Instead, a servlet waits for client’s HTTP requests and generates a response in every request.

**94. What is the difference between GenericServlet and HttpServlet ?** GenericServlet is a generalized and protocol-independent servlet that implements the Servlet and ServletConfig interfaces. Those servlets extending the GenericServlet class shall override the service method. Finally, in order to develop an HTTP servlet for use on the Web that serves requests using the HTTP protocol, your servlet must extend the HttpServlet instead. Check [Servlet examples here](http://examples.javacodegeeks.com/tag/servlet/).

**95. Explain the life cycle of a Servlet.** On every client’s request, the Servlet Engine loads the servlets and invokes its init methods, in order for the servlet to be initialized. Then, the Servlet object handles all subsequent requests coming from that client, by invoking the service method for each request separately. Finally, the servlet is removed by calling the server’s destroy method.

**96. What is the difference between doGet() and doPost() ?** doGET: The GET method appends the name-value pairs on the request’s URL. Thus, there is a limit on the number of characters and subsequently on the number of values that can be used in a client’s request. Furthermore, the values of the request are made visible and thus, sensitive information must not be passed in that way. doPOST: The POST method overcomes the limit imposed by the GET request, by sending the values of the request inside its body. Also, there is no limitations on the number of values to be sent across. Finally, the sensitive information passed through a POST request is not visible to an external client.

**97. What is meant by a Web Application ?** A Web application is a dynamic extension of a Web or application server. There are two types of web applications: presentation-oriented and service-oriented. A presentation-oriented Web application generates interactive web pages, which contain various types of markup language and dynamic content in response to requests. On the other hand, a service-oriented web application implements the endpoint of a web service. In general, a Web application can be seen as a collection of servlets installed under a specific subset of the server’s URL namespace.

**98. What is a Server Side Include (SSI) ?** Server Side Includes (SSI) is a simple interpreted server-side scripting language, used almost exclusively for the Web, and is embedded with a servlet tag. The most frequent use of SSI is to include the contents of one or more files into a Web page on a Web server. When a Web page is accessed by a browser, the Web server replaces the servlet tag in that Web page with the hyper text generated by the corresponding servlet.

**99. What is Servlet Chaining ?** Servlet Chaining is the method where the output of one servlet is sent to a second servlet. The output of the second servlet can be sent to a third servlet, and so on. The last servlet in the chain is responsible for sending the response to the client.

**100. How do you find out what client machine is making a request to your servlet ?** The ServletRequest class has functions for finding out the IP address or host name of the client machine. getRemoteAddr() gets the IP address of the client machine and getRemoteHost() gets the host name of the client machine. See example [here](http://examples.javacodegeeks.com/enterprise-java/servlet/get-client-s-address-and-hostname-in-servlet/).

**101. What is the structure of the HTTP response ?** The HTTP response consists of three parts:

* **Status Code**: describes the status of the response. It can be used to check if the request has been successfully completed. In case the request failed, the status code can be used to find out the reason behind the failure. If your servlet does not return a status code, the success status code, HttpServletResponse.SC\_OK, is returned by default.
* **HTTP Headers**: they contain more information about the response. For example, the headers may specify the date/time after which the response is considered stale, or the form of encoding used to safely transfer the entity to the user. See [how to retrieve headers in Servlet here](http://examples.javacodegeeks.com/enterprise-java/servlet/get-all-request-headers-in-servlet/).
* **Body**: it contains the content of the response. The body may contain HTML code, an image, etc. The body consists of the data bytes transmitted in an HTTP transaction message immediately following the headers.

**102. What is a cookie ? What is the difference between session and cookie ?** [A cookie](http://examples.javacodegeeks.com/core-java/net/urlconnection/get-cookies-from-http-connection/) is a bit of information that the Web server sends to the browser. The browser stores the cookies for each Web server in a local file. In a future request, the browser, along with the request, sends all stored cookies for that specific Web server.The differences between session and a cookie are the following:

* The session should work, regardless of the settings on the client browser. The client may have chosen to disable cookies. However, the sessions still work, as the client has no ability to disable them in the server side.
* The session and cookies also differ in the amount of information the can store. The HTTP session is capable of storing any Java object, while a cookie can only store String objects.

**103. Which protocol will be used by browser and servlet to communicate ?** The browser communicates with a servlet by using the HTTP protocol.

**104. What is HTTP Tunneling ?** HTTP Tunneling is a technique by which, communications performed using various network protocols are encapsulated using the HTTP or HTTPS protocols. The HTTP protocol therefore acts as a wrapper for a channel that the network protocol being tunneled uses to communicate. The masking of other protocol requests as HTTP requests is HTTP Tunneling.

**105. What’s the difference between sendRedirect and forward methods ?** The sendRedirect method creates a new request, while the forward method just forwards a request to a new target. The previous request scope objects are not available after a redirect, because it results in a new request. On the other hand, the previous request scope objects are available after forwarding. FInally, in general, the sendRedirect method is considered to be slower compare to the forward method.

**106. What is URL Encoding and URL Decoding ?** The URL encoding procedure is responsible for replacing all the spaces and every other extra special character of a URL, into their corresponding Hex representation. In correspondence, URL decoding is the exact opposite procedure.

## **JSP**

**107. What is a JSP Page ?** A Java Server Page (JSP) is a text document that contains two types of text: static data and JSP elements. Static data can be expressed in any text-based format, such as HTML or XML. JSP is a technology that mixes static content with dynamically-generated content. See [JSP example here](http://examples.javacodegeeks.com/enterprise-java/jsp/sample-jsp-java-server-page/).

**108. How are the JSP requests handled ?** On the arrival of a JSP request, the browser first requests a page with a .jsp extension. Then, the Web server reads the request and using the JSP compiler, the Web server converts the JSP page into a servlet class. Notice that the JSP file is compiled only on the first request of the page, or if the JSP file has changed.The generated servlet class is invoked, in order to handle the browser’s request. Once the execution of the request is over, the servlet sends a response back to the client. See [how to get Request parameters in a JSP](http://examples.javacodegeeks.com/enterprise-java/jsp/get-request-parameter-in-jsp-page/).

**109. What are the advantages of JSP ?** The advantages of using the JSP technology are shown below:

* JSP pages are dynamically compiled into servlets and thus, the developers can easily make updates to presentation code.
* JSP pages can be pre-compiled.
* JSP pages can be easily combined to static templates, including HTML or XML fragments, with code that generates dynamic content.
* Developers can offer customized JSP tag libraries that page authors access using an XML-like syntax.
* Developers can make logic changes at the component level, without editing the individual pages that use the application’s logic.

**110. What are Directives ? What are the different types of Directives available in JSP ?** Directives are instructions that are processed by the JSP engine, when the page is compiled to a servlet. Directives are used to set page-level instructions, insert data from external files, and specify custom tag libraries. Directives are defined between < %@ and % >.The different types of directives are shown below:

* Include directive: it is used to include a file and merges the content of the file with the current page.
* Page directive: it is used to define specific attributes in the JSP page, like error page and buffer.
* Taglib: it is used to declare a custom tag library which is used in the page.

**111. What are JSP actions ?** JSP actions use constructs in XML syntax to control the behavior of the servlet engine. JSP actions are executed when a JSP page is requested. They can be dynamically inserted into a file, re-use JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.Some of the available actions are listed below:

* jsp:include – includes a file, when the JSP page is requested.
* jsp:useBean – finds or instantiates a JavaBean.
* jsp:setProperty – sets the property of a JavaBean.
* jsp:getProperty – gets the property of a JavaBean.
* jsp:forward – forwards the requester to a new page.
* jsp:plugin – generates browser-specific code.

**112. What are Scriptlets ?** In Java Server Pages (JSP) technology, a scriptlet is a piece of Java-code embedded in a JSP page. The scriptlet is everything inside the tags. Between these tags, a user can add any valid scriplet.

**113. What are Decalarations ?** Declarations are similar to variable declarations in Java. Declarations are used to declare variables for subsequent use in expressions or scriptlets. To add a declaration, you must use the sequences to enclose your declarations.

**114. What are Expressions ?** A JSP expression is used to insert the value of a scripting language expression, converted into a string, into the data stream returned to the client, by the web server. Expressions are defined between <% = and %>tags.

**115. What is meant by implicit objects and what are they ?** JSP implicit objects are those Java objects that the JSP Container makes available to developers in each page. A developer can call them directly, without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.The following objects are considered implicit in a JSP page:

* application
* page
* request
* response
* session
* exception
* out
* config
* pageContext

Still with us? Wow, that was a huge article about different types of questions that can be used in a Java interview. If you enjoyed this, then [**subscribe to our newsletter**](http://eepurl.com/xRlkj) to enjoy weekly updates and complimentary whitepapers! Also, check out[**JCG Academy**](http://academy.javacodegeeks.com/) for more advanced training!

* 1. **Kể các collection biết. Phân biệt list và linklist. Add dãy 10 số từ 0 -10 thì nên dùng cái nào?**  
     - Collection, Set, List, Map, ArrayList, Vector, Hashtable, Hashset, HashMap  
     \*Phân biệt list và linkedlist, array, arrraylist:  
       
     *Array*  
     - Ưu điểm:  
     + Dễ cài đặt & truy cập đến các phần tử dữ liêu.  
     + Tốc độ truy cập đến 1 vị trí bất kỳ trên mảng nhanh & hiệu quả.  
       
     - Nhược điểm:  
     + Cần phải xác định số phần tử mảng trước khi sử dụng(kể cả mạng động)-> ko phù hợp với bài toán chưa biết số lượng các phần tử.  
     + Khó khăn trong khi chèn or xoá một phần tử bất kỳ. Nếu việc chèn & xoá diễn ra nhiều -> xử lý chậm.  
       
     *Linkedlist*  
     - Ưu điểm:  
     + Do DSLK dùng tới đâu cấp phát tới đó -> phù hợp với bài toán chưa xđ số phần tử cho trước.  
     + Dễ dàng trong việc xoá, chèn các phần tử trong DS(thay đổi lại địa chỉ, quan hệ của phần tử).  
       
     - Nhược điểm:  
     + Khó cài đặt và truy cập đến phần tử trong danh sách.  
     + Tốc độ truy cập đến phần tử trong DS chậm( người ta thường dùng cây quyết định để phục vụ cho việc tìm kiếm)  
       
     *ArrayList:*  
     - truy cập trực tiếp các phần tử qua set và get methods  
     - có thể thêm, tìm kiếm đối tượng dể dàng hơn array  
       
     **2. Phân biệt Vector và ArrayList?**  
     Vector có cơ chế đồng bộ hóa còn ArrayList thì khôngà ArrayList nhanh hơn vector  
       
     **3. Arraylist hơn array như thế nào?**  
     Năng động, cung cấp cơ chế chèn và tìm kiếm mạnh mẽ hơn array  
       
     **4. an Iterator là gì?**  
     Về cơ bản,bộ duyệt Iterator giúp ta duyệt các phần tử trong 1 cái danh sách, ví dụ mình có cái danh sách các tên:  
     - khi khởi tạo Iterator, sẽ có 1 con trỏ được khởi tạo  
     - hàm hasNext để kiểm tra xem mình còn duyệt được tiếp nữa không  
     - hàm next trả ra phần tử kế tiếp của bộ duyệt  
       
     Mình hoàn toàn có thể làm một class duyệt theo ý mình, ví dụ như thêm vào đó 2 biến là vị trí bắt đầu và chiều duyệt thì hoàn toàn có thể làm được duyệt nhiều kiểu  
       
     **5. Các câu hỏi khác**  
       
     + Subclass: class kế thừa tất cá các thuộc tính public và phương thức protected  
     + Break, continute:   
           Break: cham dứt hẳn  
           Continute: chấm dứt câu lệnh và quay trở lại vòng lặp  
       
     Q: Can you call one constructor from another if a class has multiple constructors?   
     A: Yes, use this() syntax.  
       
     Q: What is the default value of an object reference declared as an instance variable?   
     A: Null, unless it is defined explicitly.   
       
     Q: Is it necessary that each try block must be followed by a catch block?   
     A: It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.  
       
     Q: a private constructor?  
     A: ngăn chặn subclass và ko muốn lớp khác khởi tạo đối tượng  
       
     Q: What is difference between Path and Classpath?  
     A: Path and Classpath are operating system level environment variales. Path is defines where the system can find the executables(.exe) files and classpath is used to specify the location of .class files.  
       
     Q: Can an Interface extend another Interface?  
     A: Yes an Interface can inherit another Interface, for that matter an Interface can extend more than one Interface  
       
     Q: Em hay sử dụng những loại Collections nào và một chút đặc điểm:  
     A: Có 4 loại là Lists (lưu một danh sách các đối tượng), Sets (một danh sách các đối tượng không trùng nhau), Maps (một danh sách các đối tượng có ID), Queues (một danh sách các đối tượng được xắp sếp theo thứ tự). Mỗi loại này có nhiều lớp con.  
      Dùng gì để truy cập CSDL và thiết kế như thế nào: JDBC, Hibernate… sau đó sử dụng DAO pattern (CRUD: creat, read, insert, update, delete)  
     - Mô hình n-tier: view, bussiness logic, data  
     - Một vài loại pattern thông dụng: singleton, factory, proxy, adapter, observer …  
     - Một vài loại frameworks: Struts: MVC+ Front Controller pattern. Hibernate: object relational mapping.  
       
     Q: các tính chất của OOP  
     A: có 4 tính chất là: Tính trừu tượng, tính đa hình, tính thừa kế và tính đóng gói  
       
     Q: sự khác nhau giữa JSP và Servlet  
     A: JSP khi biên dịch thì sẽ convert sang code java, Servlet là thuần code java nên servlet sẽ nhanh hơn. Hơn nữa trong mô hình MVC thì servlet sẽ đóng vai trò controller và trả về một view JSP. Trên serlvet mà trả về code HMTL thì sẽ rất là nhàm chán nhưng JSP thì sẽ dễ hơn nhiều  
       
     Q: Khác nhau giữa equals và == trong java  
     A: equals là để so sánh giá trị giữa 2 object còn == là để so sánh reference của 2 object  
       
     Q: sư khác nhau khi anh lấy 1 phần từ từ 1 list ra và lấy phần tử tứ set ra như thế nào  
     A: lấy phần tử từ một list ra sẽ trả về 1 reference của object trong list đó, còn set thì sẽ nhận được giá trị clone (copy) của nó  
       
     A: có 2 loại là Runtime Exception (unchecked) là những ngoại lệ xảy ra trong lúc thực thi và checked exception là những ngoại lệ xảy ra trong lúc biên dịch  
       
     **6. Kể các kinh nghiệm về Java**  
       
     **7. Jad là gì? build file ở đâu?**  
       
     **8. HashMap and Map?**  
       
     **9. Phân biệt HashMap và HashTable**  
       
     **10. Phân biệt a constructor và a method?**

If there's one book that Java developers repeatedly refer to as their favorite, it's [Effective Java](http://www.amazon.com/Effective-Java-2nd-Joshua-Bloch/dp/0321356683/ref=sr_1_1?ie=UTF8&s=books&qid=1223917599&sr=1-1).

|  |  |  |
| --- | --- | --- |
|  | "It's easy to feel like the general warnings about premature optimization don't apply to you, because you just know which code is time-critical and how to make it fast. But no one can determine this without measuring before and after each attempted optimization."    |  | | --- | |  |     **Joshua Bloch**  Author, Effective Java, Second Edition |

question In the preface to the new edition of [*Effective Java*](http://www.amazon.com/Effective-Java-2nd-Joshua-Bloch/dp/0321356683/ref=sr_1_1?ie=UTF8&s=books&qid=1223917599&sr=1-1), you described moving to Google in 2004 and wrote: "I've also had the pleasure of using the Java platform to develop libraries for use within Google. Now I know what it feels like to be a user." How did your experience as a user inform the new edition of the book?

answer Well, for one thing, it reinforced my feeling that it's critical to get API designs right. For example, I found myself wanting to provide an alternative FileChannel implementation for the Google File System, but I couldn't, because FileChannel is a concrete class rather than an interface. Similarly, I wanted to implement a growable ByteBuffer. These are the sort of frustrations that ordinary developers feel every day.

question You have said that a common fault among Java developers is the natural tendency to optimize code, resulting in slower code that is needlessly complicated. Why do developers mistakenly optimize code?

answer To be a software developer, you have to be an optimist -- otherwise, it would feel like a losing battle. Generally, this is a good thing, but it has a downside: Optimism can lead to overconfidence. It's easy to feel like the general warnings about premature optimization don't apply to you, because you just know which code is time-critical and how to make it fast. But no one can determine this without measuring before and after each attempted optimization.

**Ignoring Libraries**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
| "In order to stay sane, most developers have a can-do attitude, and some take it too far. They say to themselves, 'Yes, there's a library, but I can do better.' Maybe you can, but that doesn't mean you should."    |  | | --- | |  |     **Joshua Bloch**  Author, Effective Java, Second Edition |  |

question Another fault you refer to involves developers writing their own code when perfectly good libraries exist. Why do developers do this?

answer Two reasons: By far the most common reason is that the developer doesn't know the library exists. I feel for that developer, because there are so many libraries out there that it's impossible to keep track of them all. That said, it's so hard to get some of these facilities right that it's worth making the effort to find out if a library exists.

This is particularly true where concurrency is involved. It's not uncommon for experts to spend literally months writing apparently modest concurrency utilities. When faced with a need for this sort of functionality, the wise developer will do whatever it takes to find an appropriate library. It's so easy to get nontrivial concurrent code wrong, and the resulting bugs can be nearly impossible to detect.

The second reason that developers tend to reinvent the wheel is the same reason they tend to optimize prematurely: In order to stay sane, most developers have a can-do attitude, and some take it too far. They say to themselves, "Yes, there's a library, but I can do better." Maybe you can, but that doesn't mean you should. Use the standard library unless it's profoundly unsuited to your needs.

question [George Orwell](http://en.wikipedia.org/wiki/George_Orwell) famously presented five rules of good writing -- and then added a sixth: "Break any of these rules sooner than say anything outright barbarous." [*Effective Java*](http://www.amazon.com/Effective-Java-2nd-Joshua-Bloch/dp/0321356683/ref=sr_1_1?ie=UTF8&s=books&qid=1223917599&sr=1-1)now consists of 78 items with titles consisting of short rules like "Favor generic methods" or "Consider using a custom serialized form," to choose two at random. Do you have any favorite rules? And when should developers break the rules?

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | | Joshua Bloch SigningEffective Java at Jazoon | |

answer First, I should point out that I shamelessly stole Orwell's admonition. In the introduction to Effective Java, it says, "You should not slavishly follow these rules, but violate them only occasionally and with good reason."

I can be somewhat fickle, so my favorite rules might change if you ask me on another day, but today I'll go with Item 13, "Minimize the accessibility of classes and members," and Item 15, "Minimize mutability." Both of these rules transcend any particular programming language.

The first tells you to hide information to the maximum extent feasible. This principle, due originally to [David Parnas](http://en.wikipedia.org/wiki/David_Parnas), is one of the most fundamental tenets of good programming. If you hide information, you are free to change it without risking harm to the system. Information hiding decouples the components of the system, allowing them to be developed, tested, optimized, used, understood, and modified in isolation.

Item 15 tells you to keep the state space of each object as simple as possible. If an object is immutable, it can be in only one state, and you win big. You never have to worry about what state the object is in, and you can share it freely, with no need for synchronization. If you can't make an object immutable, at least minimize the amount of mutation that is possible. This makes it easier to use the object correctly.

As an extreme example of what not to do, consider the case of [java.util.Calendar](http://docs.oracle.com/javase/6/docs/api/java/util/Calendar.html) . Very few people understand its state-space -- I certainly don't -- and it's been a constant source of bugs for years.

So when should you break the rules? To paraphrase Orwell, you should break them when they'd result in code that is "outright barbarous." As a simple example, Item 25 says, "Prefer lists to arrays." So why does the values method present on every enum type return an array? Because the most common use of this method is to iterate over the elements of its return value, and it's much cheaper to iterate over an array than anyList implementation. With the for-each construct, the code to iterate is the same either way:

|  |
| --- |
| for (Planet p : Planet.values()) |

You might think this was premature optimization on the part of the enum designers, but it wasn't. This isn't just inner loop code, it is the (inner) loop. Of course, we (the [JSR 201](http://jcp.org/en/jsr/detail?id=201) expert group) did extensive performance measurements before making this decision, and I'm satisfied that we made the right decision.

Another time that you should break the rules is when two of the rules would drive you to opposite decisions. In such cases, you're forced to violate at least one of the rules, and you have to choose which is more important, which can be very difficult.

When faced with such decisions, I typically discuss it with someone else for a sanity check. There's no shame in doing this, and it's arguably a best practice. Contrary to popular belief, software design is not -- or should not be -- a solitary occupation.

**Generics, Enums, and Annotations**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | "For generics, the sound bite is 'Don't use raw types.' ... If a library designer took the time to write a generic library, you should take advantage of it."    |  | | --- | |  |     **Joshua Bloch**  Author, Effective Java, Second Edition |

question Give us a taste of what you believe developers should know about generics, enums, and annotations.

answer Owing to space limitations, it has to be a small taste, but here goes.

For generics, the sound bite is "Don't use raw types" (Item 23). If a library designer took the time to write a generic library, you should take advantage of it. In other words, don't do this:

|  |
| --- |
| // Raw type-leads to unsafe code!  List dogs = new ArrayList(); |

Instead, do this:

|  |
| --- |
| List<Dog> dogs = new ArrayList<Dog>(); |

At first, this may seem like needless verbiage, but it's not. By telling the compiler what kind of elements the list contains, you enable it to find many errors at compile time that would otherwise cause a ClassCastException at runtime. You also eliminate ugly casts from your program.

For enums, the sound bite is "Always use enums in place of int constants" (Item 30). Enums provide so many advantages: compile-time type safety, the ability to add or remove values without breaking clients, meaningful printed values, the ability to associate methods and fields with the values, and so on. Since we have EnumSet, this advice applies equally to bit fields, which should be considered obsolete.

For annotations, the sound bite is "Don't define your own unless you have a very good reason, but do use the standard ones for your environment." For many programmers, the only ones that matter will be [@Override](http://docs.oracle.com/javase/6/docs/api/java/lang/Override.html) (Item 36) and [@SuppressWarnings](http://docs.oracle.com/javase/6/docs/api/java/lang/SuppressWarnings.html) (Item 24). Using the@Override annotation is an easy way to save yourself from errors that would otherwise be very hard to detect. Notably, it is not uncommon to accidentally overload the equals method when you intend to override it, causing subtle, pernicious errors:

|  |
| --- |
| // Broken! Unintentional overloading  public boolean equals(MyClass other) { // MyClass should be Object.  ...  } |

If you tell the compiler that you believe you're overriding a superclass method, it will inform you of your error:

|  |
| --- |
| // @Override annotation prevents broken code from compiling.  @Override public boolean equals(MyClass mc) {  ...  } |

**Best Practices for Lazy Initialization**

|  |
| --- |
|  |

question What's most important to understand about best practices for lazy initialization?

answer The single most important piece of advice is "Don't do it unless you need to." The great majority of your initialization code should look like this:

|  |
| --- |
| // Normal initialization, not lazy!  private final FieldType field = computeFieldValue(); |

If you need lazy initialization for correctness -- but not for performance -- just use a synchronized accessor. It's simple and clearly correct.

If you need better performance, your best choice depends on whether you're initializing a static field or an instance field. If it's a static field, use the lazy initialization holder class idiom:

|  |
| --- |
| // Lazy initialization holder class idiom for static fields  private static class FieldHolder {  static final FieldType field = computeFieldValue();  }  static FieldType getField() { return FieldHolder.field; } |

This idiom is almost magical. There's synchronization going on, but it's invisible. The Java Runtime Environment does it for you, behind the scenes. And many VMs actually patch the code to eliminate the synchronization once it's no longer necessary, so this idiom is extremely fast.

If you need high-performance lazy initializing of an instance field, use the double-check idiom with a volatile field. This idiom wasn't guaranteed to work until release 5.0, when the platform got a new memory model. The idiom is very fast but also complicated and delicate, so don't be tempted to modify it in any way. Just copy and paste -- normally not a good idea, but appropriate here:

|  |
| --- |
| // Double-check idiom for lazy initialization of instance fields.  private volatile FieldType field;  FieldType getField() {  FieldType result = field;  if (result == null) { // First check (no locking)  synchronized(this) {  result = field;  if (result == null) // Second check (with locking)  field = result = computeFieldValue();  }  }  return result;  } |

For more information on this topic, see Item 71.

question Tell us something that you are really proud of in the second edition of Effective Java.

answer I'm proud that I was able to preserve the feel of the book, even as the language grew. One reason the first edition was so successful is that it was small and approachable. The addition of all the new language and library features made the platform larger and more complex. Inevitably, the book grew (from 57 items to 78), but I tried really hard to keep it short and clear. Based on early feedback, I think I was largely successful.

**The Strangest Thing About the Java Platform**

|  |
| --- |
|  |

question After having seven more years to reflect on the development of the Java platform, what's the strangest thing you can say about it?

answer Ooh, good question... I'm going to say that the strangest thing about the Java platform is that the byte type is signed. I've never heard an explanation for this. It's quite counterintuitive and causes all sorts of errors. For example, what do you think this program does?

|  |
| --- |
| class Delight {  public static void main(String[] args) {  for (byte b = Byte.MIN\_VALUE;  b < Byte.MAX\_VALUE; b++) {  if (b == 0x90)  System.out.print("Joy!");  }  }  } |

If you have any doubts, run it. If you need an explanation for the behavior, find a copy of [*Java Puzzlers*](http://www.amazon.com/Java-TM-Puzzlers-Pitfalls-Corner/dp/032133678X/ref=sr_1_1?ie=UTF8&s=books&qid=1224040620&sr=1-1)and look at Puzzle 24.

**Unit Tests Are Essential**

|  |
| --- |
|  |

question [Java Champion Dr. Heinz Kabutz](http://www.oracle.com/technetwork/articles/javase/kabutz-qa-136652.html) finds that failure to unit test is a big problem among Java developers. He reports, "At conferences, I ask, 'How many of you have unit tests for your code?' Almost no one raises their hands -- and these are experienced professionals." Your reaction?

answer I'm really sorry to hear it. Unit tests are essential! If you don't have them, you don't know whether your code works. Having a good set of unit tests gives you much more confidence that your code works in the first place and that you don't introduce bugs as you maintain it. Inevitably, you will introduce bugs, but your unit tests will often let you find the bugs as soon as you introduce them, so you can fix them before they cause any damage.

Note, however, that unit tests are not sufficient to ensure that your code works. You need to understand the code and to prove to yourself that it works. And you need to have someone else review it. When it comes to code, two heads really are better than one. For a more detailed discussion of this topic, see my [binary search article](http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html).

**ReadResolve Isn't All It's Cracked up to Be**

|  |
| --- |
|  |

question When you revised Effective Java, you said that you had to critically examine everything in the first edition. Was there anything you originally wrote that you realized was misguided?

answer Yes. For one thing, I used to think that a readResolve method reliably guaranteed that a singleton remained a singleton in the face of serialization and deserialization. It turns out that this isn't quite true. (See Item 77.)

Luckily, there's a nice solution: implement your serializable singleton as a single-element enum. Also, I discovered various minor embarrassments as I went over the first edition with a fine-toothed comb, in the form of typos that eluded detection for seven years. Luckily, I don't remember any of them right now.

**Become a Better Writer**

|  |
| --- |
|  |

question You have said that developers should read Strunk and White's book [*The Elements of Style*](http://www.amazon.com/Elements-Style-Fourth-William-Strunk/dp/020530902X)because being a better writer will make you a better developer. Can you explain why this is so?

answer I believe that reading Strunk and White will make you a better developer because good programming and good writing are both about clarity and economy of expression. You can't write good code or good prose unless you understand what it is you're trying to say. Many of Strunk and White's admonitions have direct analogues for software. For example, Strunk and White say, "Omit needless words!" where Andy Hunt and Dave Thomas ( ["The Pragmatic Programmers"](http://www.pragprog.com/the-pragmatic-programmer" \t "_blank)) say, "Don't repeat yourself." Strunk and White say, "Revise and Rewrite," where [Martin Fowler](http://martinfowler.com/)says, "refactor." And the list goes on.

In fact, much of Strunk and White's advice doesn't even have to be paraphrased to apply to software. Here are a few examples, taken verbatim from the table of contents of The Elements of Style:

* Choose a Suitable Design and Hold to It
* Be Clear
* Do Not Take Shortcuts at the Cost of Clarity

question Which new principles in Effective Java might be integrated as best practices into the [NetBeans IDE](http://www.netbeans.org/)?

answer Many of the rules in the first edition have been captured as "inspections" in IDEs such as NetBeans, IntelliJ, and Eclipse. Similarly, the FindBugs tool does a good job of automating these checks. I suspect that many of the new rules will find their way into these tools.

As a simple example, I know that IDEs already check for the presence of the @Override annotation and warn you when you've forgotten it.

**How Generics Worked Out**

|  |
| --- |
|  |

question [In 2003, you commented](http://www.oracle.com/technetwork/articles/java/bloch-effective-08-qa-140880.html) that if any of the changes in J2SE 5.0 might present particular challenges for Java developers, it would be generics, because developers would "have to get used to providing additional information in declarations." How do you think generics have worked out?

answer Merely adding type information to declarations hasn't proven to be all that difficult, but other aspects of generics have proven far more challenging. For instance, we vastly underestimated the complexity of wildcard types. They were added late in the development cycle, and we didn't appreciate all of the subtleties they introduced.

Generics certainly improved the type safety and expressiveness of the language, and I'm very glad they were added. But they haven't been an unqualified success. You only have to peruse Angelika Langer's 513-page [Java Generics FAQs](http://www.angelikalanger.com/GenericsFAQ/JavaGenericsFAQ.html) to appreciate this. So I wish we'd been able to simplify the design.

I also wish we'd found some way to make generics work with primitive types, so that we could have avoided autoboxing and auto-unboxing, which have proven bug prone.

What do you think this program prints? If you have any doubts, run it:

|  |
| --- |
| public class Unbelievable {  static Integer i;  public static void main(String[] args) {  if (i == 42)  System.out.println("Unbelievable");  }  } |

**The Future of the Java Platform**

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | "I'm convinced that the Java programming language has used up its complexity budget with the changes introduced in release 5.0. It would be a huge mistake to add any new language features that significantly increased the complexity of the language."    |  | | --- | |  |     **Joshua Bloch**  Author, Effective Java, Second Edition |

question Do you want to say anything about the future of the Java platform generally and closures in particular?

answer It's difficult to predict the future of the platform. Things have been moving a bit slowly since the release of[Java SE 6](http://www.oracle.com/technetwork/java/javase/overview/index.html), and no one is quite sure what is coming in Java SE 7, or when it's coming.

I'm convinced that the Java programming language has used up its complexity budget with the changes introduced in release 5.0. It would be a huge mistake to add any new language features that significantly increased the complexity of the language. I would place [BGGA Closures](http://www.javac.info/) squarely in this category. For more on this topic, you can take a look at the [Closures Controversy talk](http://www.parleys.com/display/PARLEYS/Home#slide=1;title=The%20Closures%20Controversy;talk=5210267).

A modest collection of minor language changes could improve the language, but it would have to be done with the utmost restraint.

**On Being Zany and Goofy While Developing**

|  |
| --- |
|  |

question You have a long history of emphasizing fun in everything you do. At your defense of your Ph.D. dissertation, you answered a planted question with a rap song backed by a recorded rhythm track on a hidden tape recorder. And you and Neal Gafter did a "Java Puzzlers" routine in which you donned mechanics' overalls and called yourselves "Click and Hack, the Type-It Brothers," after the radio show [*Car Talk*](http://www.cartalk.com/). What can you say about the importance of zaniness, goofiness, and joy in the process of developing?

answer It's a huge part of who I am and what I do. As I always say, computer science is an immature discipline, and I aim to keep it that way. I do much better work when I'm having fun and pursuing my passion. I'm deeply thankful that I've been able to spend so much of my academic and professional careers doing exactly that.

**Advice for New Developers**

|  |
| --- |
|  |

question Effective Java is a book written for experienced Java developers. Do you have any advice for how a computer science major in college might benefit from the book? Or an experienced developer moving from another language to Java?

answer Many programmers find it useful to keep a copy at their desk so they can look at the code examples and such while they work. Also, I've seen people use the book's "Item" references in comments explaining their design decisions. Needless to say, I'm honored that they've done so.

As for programmers moving from another language, I think they'd do well to read a quick introduction to Java before tackling my book. Peter Sestoft's book [*Java Precisely*](http://www.amazon.com/Java-Precisely-2nd-Peter-Sestoft/dp/0262693259/ref=sr_1_2?ie=UTF8&s=books&qid=1224723136&sr=1-2)would be an excellent choice.

# Regular Expression

## 1. Tổng quát

Trong cuộc sống hằng ngày, con người chúng ta diễn đạt một ý niệm hay một việc làm qua lời nói, cử chỉ hay bằng chữ viết. Bởi vì con người chúng ta có 5 giác quan khác nhau và những giác quan đó giúp chúng ta hiểu được vấn đề qua sự nhìn, nghe, ngửi, nếm, cảm thấy. Nói một cách tổng quát, trong xã hội văn minh thì chúng ta làm việc và tiếp xúc với nhau qua cách dùng ngôn ngữ. Như người Việt thì dùng tiếng Việt để trao đổi tin tức hay diễn đạt một vấn đề. Nhưng máy tính thì không có những giác quan như con người chúng ta, vậy làm sao chúng ta có thể trao đổi thông tin với vi tính một cách đơn giản nhưng hiệu quả?

Theo toán học, thành ngữ là một tập hợp của chữ viết. Mỗi chữ viết bao gồm một chuỗi ký tự xác định (a language is a set of finite string; a string is a finite sequence of symbols -Appel 1998). Cho nên nếu con người có thể diễn tả một tập hợp chữ viết bằng một phương pháp nào đó thì nó tương đương với việc diễn tả một thành ngữ.

Cấu trúc đại diện hay Regular Expression (Regexp) là một dạng viết ngắn gọn để mô tả một tập hợp chữ viết và có thể tạm dịch là một phương pháp dùng để mô tả hay diễn đạt thành ngữ một cách ngắn gọn và hiệu quả. Ví dụ như để diễn tả một tập hợp chữ chỉ chứa những chữ cái và chỉ chữ cái mà thôi (a set of string contains only alphabet), thì Regexp có thể diễn tả câu nói trên ngắn gọn như sau: [a-zA-Z]\*

Mục đích của bài viết này là giới thiệu với người đọc về Regexp & những ứng dụng của nó trong việc lập trình hằng ngày. Phần đầu chúng ta đã giới thiệu sơ lược về Regexp. Phần 2 sẽ nói về những quy luật tạo dựng nên một Regexp. Phần 3 sẽ trình bày về những ứng dụng của Regexp, phần 4 chúng ta đi vào phần thực hành, và phần 5 chúng ta sẽ tóm tắt bài viết.

## 2. Cấu Trúc

Trước khi nói về những thể loại của Regexp, chúng ta cần phải thống nhất vài điều

|  |
| --- |
| Giả định thành ngữ là một tập hợp của chữ viết bao gồm chữ cái, số, dấu ký tự. Từ đây về sau chúng ta có thể dùng tập hợp của chữ viết để diễn đạt một thành ngữ. |
| Trong tân toán học, A OR B tương đương với A hoặc B |
| A AND B tương đương vói A và B |
| Chuỗi kí tự trống hay một tập hợp trống là một tập hợp không có kí tự nào hết (empty string) |
| {...} được gọi là tập hợp hay a set |
| Thành ngử là một tập hợp con của ngôn ngữ (language) |
| Theo ký hiệu hiệu toán học ngôn ngữ là L, thì thành ngữ là L(E) |

Cơ bản Regexp có 5 thể loại (Appel 2000)

Symbol (kí hiệu -Tự điển Anh Việt-Nhà Xuất Bản Khoa Học Hà Hội, 1994)

Dùng để diễn đạt một thành ngữ chỉ chứa duy nhất ký tự đó mà thôi. Thể loại này thường được dùng để diễn đạt những chữ viết bình thường hay một nhóm chữ đã được xác định.

Ví dụ: regex "a" diễn tả thành ngữ chỉ chứa một chữ cái là “a”. Hay regex "abc" diễn tả thành ngữ chứa "abc"

Alternation (thay thế)

Dùng để diễn đạt tính chất OR (cái này hoặc cái kia). Dấu hiệu | được dùng để diễn tả alternation.

Ví dụ: Giả sử Regexp A nhận ký tự a và regex B nhận ký tự b, thì alternation của A và B hay A | B sẽ là một Regexp mới và nhận a hoặc b. Nói cách khác A | B sẽ chấp nhận một thành ngữ chứa ký tự a hoặc b. Viết ngắn gọn A | B mô tả thành ngữ {"a", b"}

Concatenation(kết hợp)

Dùng để diễn đạt tính chất AND. Dấu chấm (.) được dùng để diễn tả AND Ví dụ: Giả sử A and B là 2 Regexp, thì concatenation của A và B hay là A.B (A chấm B). Ở đây concatenation là dấu chấm (.).

Nếu regex A nhận ký tự a và regex B nhận ký tự b, thì concatenation của A và B hay A.B sẽ là một Regexp mới và nhận a và b.

Nói cách khác A.B sẽ chấp nhận một thành ngữ chứa cả hai ký tự a và b. Viết ngắn gọn AB mô tả thành ngữ {"ab"}. Để đơn giản quá cách viết, A.B có thể viết không có dấu chấm như sau AB.

Epsilon

Dùng để diễn đạt chuỗi kí tự trống (coi định nghĩa d). Dấu є (epsilon) được dùng để diễn tả chuỗi kí tự trống

Ví dụ: ("ab" | e) mô tả thành ngữ {"", "ab"}

Repetition (lặp đi lặp lại)

Dùng để diễn đạt sự lặp đi lặp lại của ký tự. Có vài dấu hiệu dùng để diễn tả sự lặp lại. Dấu ngôi sao (\*) dùng để diễn tả 0 đến nhiều vô kể hay 0 --> vô cùng. Dấu cộng (+) dùng để diễn tả 1 đến nhiều vô kể hay 1 --> vô cùng. Dấu hỏi (?) dùng để diễn tả 0 đến 1.

Ví dụ:

(0 | 1)\* diễn tả số nhị phân (binary number) như {,0,01,00,111,01010, ......} (hàng đầu tiên trước dấu phẩy là chuỗi kí tự trống) (a | b | c)+ diễn tả từ gồm tập hợp những chữ {a,b,c,ab,abc, .....}

## 3. Ứng dụng

Regex rất quan trọng và được ứng dụng trong việc phân tích từ vựng (lexical analysis). Lexical analyzer (phần mềm phân tích từ vựng) là một phần trong trình biên dịch (compiler), hệ thống quản trị cơ sở dữ liệu (DBMS), phần mềm phân tích văn bản (text analyzer), máy dịch (machine translation) vân vân. Ngoài ra hiểu biết về Regexp không những sẽ giúp lập trình viên rất nhiều trong việc hiểu biết cấu trúc bên trong của compiler hay DBMS nhưng còn có thể giúp giải quyết những vấn đề gặp phải trong việc lập trình hằng ngày.

Ví dụ như gần đây có rất nhiều hãng nhỏ mở ra chủ yếu là viết chương trình chống lại thư rác (SPAM), làm cách nào để phân biệt được thư rác và thư đúng, Regexp đóng một vai trò quan trọng trong việc chọn lọc dựa trên những mô hình (patterns), những ứng dụng phổ thông hằng ngày như kiểm tra địa chỉ email, địa chỉ IP vân vân.ân.

## 4. Thực hành

Phần này chúng ta bắt đầu thực tập dùng Regexp trong Java. Trong Java từ JDK 1.4 trở đi, nó có kèm thèo gói (package) java.util.regex cho phép lập trình viên tạo Regexp dùng Java. Trong gói đó có 3 lớp gọi là Pattern, Matcher & môt lớp Exception sẽ không kề cập đến.

Lớp Pattern dùng nhận Regexp vào và kiểm tra những chữ (string) cho vào dựa trên Regexp đã tạo ra. Thông thường để nhận một Regexp, thì dùng phương thức compile trong lớp Pattern (sẽ đề cập chi tiết)

Lớp Matcher dùng để so sánh, tìm kiếm những chữ đưa vào dựa trên Regexp đã tạo ra.

Mặc dù trong lý thuyết thì Regexp chỉ có 5 thể loại nhưng trong thực tế để đơn giản hóa việc viết Regexp, những thành ngữ lập trình như Perl còn cho ra hàng loạt ký tự để mô tả Regexp. Trong bài này chúng ta chỉ đề cập đến các ký tự dùng phổ biến trong Java và các thành ngữ lập trình khác.

Ví dụ: Nếu dựa trên công thức trong phần 2, thì để diễn tả một tập hợp chữ cái và chỉ chữ cái mà thôi thì phải viết như sau:

(a | b | c | d | e....z | A | B ..... \ Z)\* (Lưu ý vì người viết dùng .... để thay thế cho những chữ trong 24 chữ cái)

Trong thực tế nếu mà chữ xếp liền mạch như vậy thì có thể viết ngắn gọn như sau: [a-zA-Z]\* dấu gạch ngang (-) dùng để mô tả sự liên tục và [...] mô tả OR

ví dụ: [abc] tương đương với [a|b|c]

Java và Perl còn có ký tự ngắn hơn như [\w] tương đương [a-zA-Z] vân vân. Độc giả nên tham khảo thêm Java API để biết thêm về những ký tự khác

Sau đây là vài ví dụ tạo Regexp dùng Java.

1. Tạo Regexp để kiểm tra tên người.,

Tên người thì bao gồm chữ cái và kí tự trống (space) với một điều kiện là tên bắt đầu phải là chữ chứ không là khoảng trống.

Ví dụ như john smith, hay john doe.

a. Trước tiên tạo một Regexp

String regexp = "[a-zA-Z]+[a-zA-Z ]\*";

// vì không cho phép có khoảng trống, nên bắt đầu phải là chữ [a-zA-Z]+

// và theo sau đó có thể là chữ hoặc khoảng trống (space) [a-zA-Z ] (lưu ý khoảng trống sau cùng trong ngoặc)

Có rất nhiều cách để tạo regexp, khuyến khích độc giả nên tham khảo Java API và dùng những cách khác để tạo ra Regexp có chức năng giống như vây

b. Sau đó tạo một pattern và nhận regexp vào để so sánh

Pattern pattern = Pattern.compile(regexp);

c. Thử kiểm tra vài tên

String test1 = " John Doe"; //invalid

String test2 = "John Doe"; //valid

String test3 = "Jhon333"; //invalid

boolean result1 = pattern.matcher(test1).matches(); //false

boolean result2 = pattern.matcher(test2).matches(); //true

boolean result3 = pattern.matcher(test3).matches();//false

Chương trình Java

-------------------------------------------------------------------------------------------

String test1 = " John Doe"; //invalid

String test2 = "John Doe"; //valid

String test3 = "Jhon333"; //invalid

String regexp = "[a-zA-Z]+[a-zA-Z ]\*";

Pattern pattern = Pattern.compile(regexp);

boolean result1 = pattern.matcher(test1).matches(); //false

boolean result2 = pattern.matcher(test2).matches(); //true

boolean result3 = pattern.matcher(test3).matches();//false

System.out.println(result1 + ":" + result2 + ":" + result3);

------------------------------------------------------------

2. Kiểm tra địa chỉ của giao thức Internet Protocol (IP Address), Địa chỉ của giao thức Internet Protocol quy định như sau:

|  |
| --- |
| Chỉ chứa số nguyên, dấu chấm, và dấu \* |
| Bắt đầu phải là \* hay số nguyên |
| Tối đa có 3 dấu chấm và mỗi một khúc thì chỉ có từ 1 đến 3 số hay một dấu \* |

ví dụ như những địa chỉ IP sau thì đúng theo quy định.

\*

192.\*

192.1.4.\*

192.\*.3.2

Phương pháp thông thường nên chia vấn đề nhỏ ra (Divide & Conquer – Chia để trị) để dễ giải quyết.

Chúng ta nhận thấy rằng trường hợp đầu tiên hơi đặc biệt và không giống những pattern khác, nên tách nó ra thành một Regexp riêng

String regex1 = "\*"; // diễn tả dấu \*

Bây giờ thì phân tích 3 trường hợp còn lại. Vì bắt đầu là số nguyên nên chúng ra có thể dùng [1-9] thể diễn tả số nguyên từ 1 đến 9. Nếu để ý kỹ chúng ta sẽ thấy có sự giống nhau trong mỗi khúc. Mỗi khúc bao gồm 1 đến 3 số nguyên hay 1 dấu \*. Địa chỉ IP thì coi tất cả 4 khúc. Cho nên chúng ta có thể diễn tả địa chỉ IP như sau:

String ip "\*" | "([0-9\*]{1,3}(.)){1,3}[0-9\*]{0,3}";

|  |  |
| --- | --- |
| [Warning] | **Warning** |
| Lưu ý: Regexp này còn một chỗ hở là nó cho nhận chữ có 3 dấu \*. Như vậy thì không đúng với điều kiện 3. Hy vọng độc giả qua bài này có thể khắc phục lỗi này. | |

Chương trình Java sẽ giống như sau:

-----------------------------------------------------------------------------------

Pattern pattern1 = Pattern.compile("[\*]"); //first case

Pattern pattern2 = Pattern.compile("([0-9\*]{1,3}(.)){1,3}[0-9\*]{0,3}");

//nhập pattern1 & pattern2

public boolean is IP(String s)

{

return ( pattern1.matcher(s).matches() || pattern2.matcher(s).matches());

}

String test1 = "192.1.4.6";

String test2 = "\*";

String test3 = "192.\*.4.6";

String test4 = "192.1.4.\*";

System.out.println(pattern2.matcher(test1).matches());

System.out.println(pattern1.matcher(test2).matches());

System.out.println(pattern2.matcher(test3).matches());

System.out.println(pattern2.matcher(test4).matches());

----------------------------------------------------------------------------------

3. Kiểm tra địa chỉ email (bài này để độc giả tự giải ), Quy tắc tạo địa chỉ email như sau:

|  |
| --- |
| Bắt đầu là chữ cái |
| Không phân biệt chữ hoa hay chữ thường (upper case hay lower case) |
| Email có thể có bao gồm chữ là số, dấu chấm, và @(dung34343.mai@dung.com) |
| Email phải kết thúc bằng những chữ cái (gov, org, vn, uk, com, net ....) |

## 5. Kết Luận

Trong bài này chúng ta đã trình bày sơ lược về regular expression. Hy vọng rằng qua bài giới thiệu này độc giả có được một kiến thức cơ bản về regular expression. Để nắm vững cách xử dụng regexp thì cần phải thực hành nhiều. Những ví dụ cho thấy regular expression đóng vai trò quan trọng trong việc phân tích chữ và là nền tảng của nhiều phần mềm. Tuy nhiên Regexp giúp compiler hay parser tách một từ ngữ ra thành những ký tự. Để compiler hay parser có thể phân tích những ký hiệu đó dựa trên văn phạm (grammar) và ý nghĩa (semantics) của thành ngữ thì những ký tự đó phải có một cấu trúc dữ liệu rõ ràng (data structure), và trong bài tới, chúng ta sẽ đề cập đến một cơ chế giúp chuyển đổi những ký tự từ Regexp thành một cấu trúc rõ ràng. Cơ chế đó gọi là Deterministic Finite Automata (DFA) and Non-deterministic Finite Automata (NFA).

## 6. Tham khảo

Cain, A. J. http://www.cenius.net/refer/display.php?ArticleID=kleeneclosure. Lấy từ trên mạng vào ngày 27 tháng 4, 2004

Appel, A. (1998). Modern Compiler Implementation in Java. UK: Cambridge University Press.

(1994). Tự điển Anh-Việt. Nhà Xuất Bản Khoa Học Hà Hội

1- Biểu thức chính quy (Regular expression) [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

1.1- Tổng quan [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

Một biểu thức chính quy (Regular expressions) định nghĩa một khuôn mẫu (pattern) tìm kiếm chuỗi. Nó có thể được sử dụng tìm kiếm, sửa đổi, và thao tác trên văn bản. Khuôn mẫu được định nghĩa bởi biểu thức chính quy có thể khớp một hoặc một vài lần, hoặc không khớp với một văn bản cho trước.  
  
Viết tắt của biểu thức chính quy là regex

1.2- Hỗ trợ các ngôn ngữ [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

Biểu thức chính quy (Regular expression) được hỗ trợ bởi hầu hết các ngôn ngữ lập trình, ví dụ, Java, Perl, Groovy, vv Thật không may mỗi ngôn ngữ hỗ trợ biểu thức thông thường hơi khác nhau.

2- Quy tắc viết biểu thức chính quy [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

|  |  |  |
| --- | --- | --- |
| No | Regular Expression | Mô tả |
| 1 | . | Khớp (match) với bất kỳ ký tự nào |
| 2 | ^regex | Biểu thức chính quy phải  khớp tại điểm bắt đầu |
| 3 | regex$ | Biểu thức chính quy phải khớp ở cuối dòng. |
| 4 | [abc] | Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c. |
| 5 | [abc][vz] | Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c theo sau là v hay z. |
| 6 | [^abc] | Khi dấu ^ xuất hiện như là nhân vật đầu tiên trong dấu ngoặc vuông, nó phủ nhận mô hình.  Điều này có thể khớp với bất kỳ ký tự nào ngoại trừ a hoặc b hoặc c. |
| 7 | [a-d1-7] | Phạm vi: phù hợp với một chuỗi giữa a và điểm d và con số từ 1 đến 7. |
| 8 | X|Z | Tìm X hoặc Z. |
| 9 | XZ | Tìm X và theo sau là Z. |
| 10 | $ | Kiểm tra kết thúc dòng. |
|  | | |
| 11 | \d | Số bất kỳ, viết ngắn gọn cho [0-9] |
| 12 | \D | Ký tự không phải là số, viết ngắn gon cho [^0-9] |
| 13 | \s | Ký tự khoảng trắng, viết ngắn gọn cho [ \t\n\x0b\r\f] |
| 14 | \S | Ký tự không phải khoản trắng, viết ngắn gọn cho [^\s] |
| 15 | \w | Ký tự chữ, viết ngắn gọn cho [a-zA-Z\_0-9] |
| 16 | \W | Ký tự không phải chữ, viết ngắn gọn cho [^\w] |
| 17 | \S+ | Một số ký tự không phải khoảng trắng (Một hoặc nhiều) |
| 18 | \b | Ký tự thuộc a-z hoặc A-Z hoặc 0-9 hoặc \_, viết ngắn gọn cho [a-zA-Z0-9\_]. |
|  | | |
| 19 | \* | Xuất hiện 0 hoặc nhiều lần, viết ngắn gọn cho {0,} |
| 20 | + | Xuất hiện 1 hoặc nhiều lần, viết ngắn gọn cho {1,} |
| 21 | ? | Xuất hiện 0 hoặc 1 lần, ? viết ngắn gọn cho {0,1}. |
| 22 | {X} | Xuất hiện X lần, {} |
| 23 | {X,Y} | Xuất hiện trong khoảng X tới Y lần. |
| 24 | \*? | \* có nghĩa là xuất hiện 0 hoặc nhiều lần, thêm ? phía sau nghĩa là tìm kiếm khớp nhỏ nhất. |

3- Các ký tự đặc biệt trong Java Regex (Special characters) [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

Một số ký tự đặc biệt trong **Java Regex**:

\.[{(\*+?^$|

Những ký tự liệt kê ở trên là các ký tự đặc biệt. Trong Java Regex bạn muốn nó hiểu các ký tự đó theo cách thông thường bạn cần thêm dấu \ ở phía trước.  
  
Chẳng hạn ký tự chấm . java regex đang hiểu là một ký tự bất kỳ, nếu bạn muốn nó hiểu là một ký tự chấm thông thường, cần phải có dấu \ phía trước.

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5 | // Mẫu regex mô tả một ký tự bất kỳ.  String regex = ".";    // Mẫu regex mô tả  ký tự dấu chấm.  String regex = "\\."; |

4- Sử dụng String.matches(String) [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

* **Class String**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5 | ...    // Kiểm tra đối tượng toàn bộ String có khớp với regex hay không.  public boolean matches(String regex)  .. |

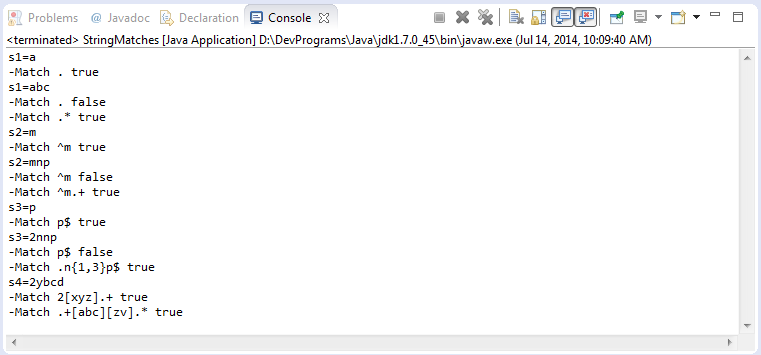
Sử dụng  method String.matches(String regex) cho phép bạn kiểm tra toàn bộ String có khớp với regex không. Đây là một cách thông dụng nhất. Hãy xem các ví dụ:

* **StringMatches.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103 | package org.o7planning.tutorial.regex.stringmatches;    public class StringMatches {     public static void main(String[] args) {         String s1 = "a";       System.out.println("s1=" + s1);       // Kiểm tra toàn bộ s1       // Khớp với bất kỳ ký tự nào       // Quy tắc: .       // ==> true       boolean match = s1.matches(".");       System.out.println("-Match . " + match);         s1 = "abc";       System.out.println("s1=" + s1);       // Kiểm tra toàn bộ s1       // Khớp với bất kỳ ký tự nào.       // ==> false  (Rõ ràng, chuỗi 3 ký tự sao khớp với 1 ký tự bất kỳ?)       match = s1.matches(".");       System.out.println("-Match . " + match);         // Kiểm tra toàn bộ s1       // Khớp với bất kỳ ký tự nào 0 hoặc nhiều lần       // Kết hợp các quy tắc: . và \*       // ==> true       match = s1.matches(".\*");       System.out.println("-Match .\* " + match);         String s2 = "m";       System.out.println("s2=" + s2);       // Kiểm tra toàn bộ s2       // Bắt đầu bởi m       // Quy tắc ^       // true       match = s2.matches("^m");       System.out.println("-Match ^m " + match);         s2 = "mnp";       System.out.println("s2=" + s2);       // Kiểm tra toàn bộ s2       // Bắt đầu bởi m       // Quy tắc ^       // ==> false  (Rõ ràng, chuỗi 3 ký tự sao khớp với 1 ký tự bất kỳ bắt đầu bởi m)       match = s2.matches("^m");       System.out.println("-Match ^m " + match);         // Bắt đầu bởi m       // Sau đó là ký tự bất kỳ, xuất hiện 1 hoặc nhiều lần.       // Quy tắc ^ và . và +       // true       match = s2.matches("^m.+");       System.out.println("-Match ^m.+ " + match);         String s3 = "p";       System.out.println("s3=" + s3);       // Kiểm tra s3 kết thúc bằng p       // Quy tắc $       // true       match = s3.matches("p$");       System.out.println("-Match p$ " + match);         s3 = "2nnp";       System.out.println("s3=" + s3);       // Kiểm tra toàn bộ s3       // Kết thúc bằng p       // ==> false  (Rõ ràng, chuỗi 4 ký tự sao khớp với 1 ký tự p cuối cùng)       match = s3.matches("p$");       System.out.println("-Match p$ " + match);         // Kiểm tra toàn bộ s3       // Ký tự bất kỳ xuất hiện 1 lần: .       // tiếp theo là n, xuất hiện 1 hoặc tối đa 3 lần.       // Kết thúc bởi p: p$       // Kết hợp các quy tắc: . , {X,Y}, $       // true       match = s3.matches(".n{1,3}p$");       System.out.println("-Match .n{1,3}p$ " + match);         String s4 = "2ybcd";       System.out.println("s4=" + s4);       // Bắt đầu là 2       // Tiếp theo x hoặc y hoặc z       // Tiếp theo bất kỳ 1 hoặc nhiểu lần.       // Kết hợp các quy tắc: [abc] , . , +       // true       match = s4.matches("2[xyz].+");         System.out.println("-Match 2[xyz].+ " + match);         String s5 = "2bkbv";       // Bắt đầu là bất kỳ, 1 hoặc nhiểu lần       // Tiếp theo a hoặc b, hoặc c: [abc]       // Tiếp theo z hoặc v: [zv]       // Tiếp theo bất kỳ       // true       match = s5.matches(".+[abc][zv].\*");         System.out.println("-Match .+[abc][zv].\* " + match);   }    } |

Kết quả chạy ví dụ:

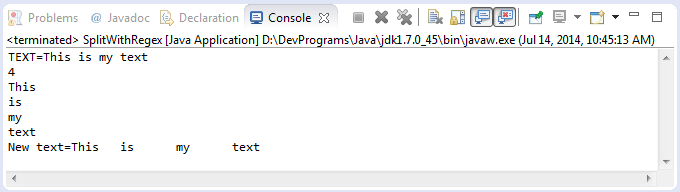


* **SplitWithRegex.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package org.o7planning.tutorial.regex.stringmatches;    public class SplitWithRegex {       public static final String TEXT = "This is my text";       public static void main(String[] args) {         System.out.println("TEXT=" + TEXT);         // Khoảng trắng xuất hiện 1 hoặc nhiều lần.         // Các ký tự khoảng trắng: \t\n\x0b\r\f         // Kết hợp quy tắc: \s và +         String regex = "\\s+";         String[] splitString = TEXT.split(regex);         // 4         System.out.println(splitString.length);           for (String string : splitString) {             System.out.println(string);         }           // Thay thế tất cả các khoảng trắng với ký tự tab.         String newText = TEXT.replaceAll("\\s+", "\t");         System.out.println("New text=" + newText);     }  } |

Kết quả chạy ví dụ:

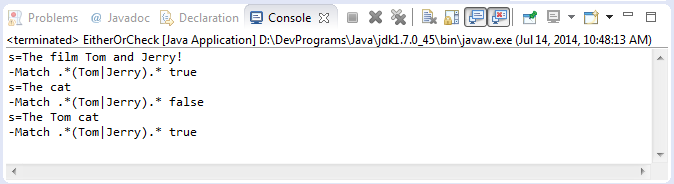


**EitherOrCheck.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package org.o7planning.tutorial.regex.stringmatches;    public class EitherOrCheck {       public static void main(String[] args) {           String s = "The film Tom and Jerry!";         // Kiểm tra toàn bộ s         // Bắt đầu bởi ký tự bất kỳ xuất hiện 0 hoặc nhiều lần         // Tiếp theo là từ Tom hoặc Jerry         // Kết thúc bởi ký tự bất kỳ xuất hiện 0 hoặc nhiều lần         // Kết hợp các quy tắc: ., \*, X|Z         // true         boolean match = s.matches(".\*(Tom|Jerry).\*");         System.out.println("s=" + s);         System.out.println("-Match .\*(Tom|Jerry).\* " + match);           s = "The cat";         // false         match = s.matches(".\*(Tom|Jerry).\*");         System.out.println("s=" + s);         System.out.println("-Match .\*(Tom|Jerry).\* " + match);           s = "The Tom cat";         // true         match = s.matches(".\*(Tom|Jerry).\*");         System.out.println("s=" + s);         System.out.println("-Match .\*(Tom|Jerry).\* " + match);     }    } |

Kết quả chạy ví dụ:



5- Sử dụng Pattern và Matcher [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

1. **Pattern**là một đối tượng mẫu, một phiên bản biên dịch của biểu thức chính quy. Nó không có cấu tử public, và chúng ta sẽ sử dụng method tĩnh compile(String) để tạo đối tượng, với tham số là biểu thức chính quy.  
  
2. **Matcher**là một phương tiện để khớp với String dữ liệu vào với đối tượng **Pattern**đã tạo trước đó. Class này không có cấu tử public, và chúng ta lấy đối tượng này thông qua method matcher(String) của đối tượng pattern. Với tham số String là văn bản cần kiểm tra.  
  
3. **PatternSyntaxException**sẽ bị ném ra nếu biểu thức chính quy có ngữ pháp không chính xác.

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | String regex= ".xx.";  // Tạo đối tượng Pattern thông qua method tĩnh.  Pattern pattern = Pattern.compile(regex);  // Lấy ra đối tượng Matcher  Matcher matcher = pattern.matcher("MxxY");    boolean match = matcher.matches();    System.out.println("Match "+ match); |

* **Class Pattern:**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

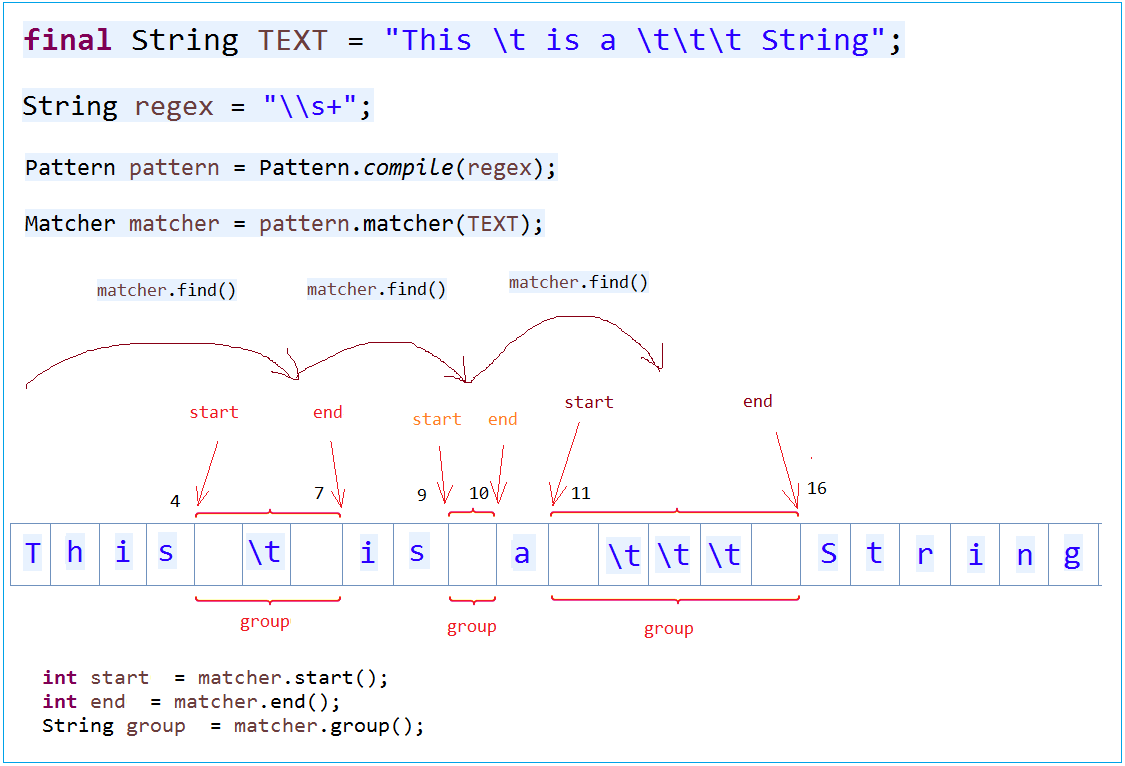
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public static Pattern compile(String regex, int flags) ;    public static Pattern compile(String regex);    public Matcher matcher(CharSequence input);    public static boolean matches(String regex, CharSequence input); |

* **Class Matcher:**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | public int start()    public int start(int group)    public int end()    public int end(int group)    public String group()    public String group(int group)    public String group(String name)    public int groupCount()    public boolean matches()    public boolean lookingAt()    public boolean find() |

Đây là một ví dụ sử dụng Matcher và method find() để tìm kiếm các chuỗi con khớp với biểu thức chính quy.

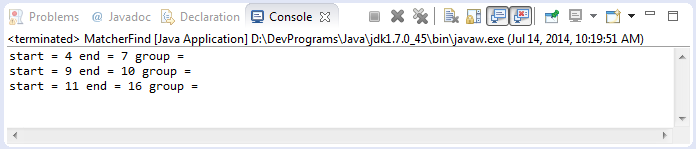


* **MatcherFind.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | package org.o7planning.tutorial.regex;    import java.util.regex.Matcher;  import java.util.regex.Pattern;    public class MatcherFind {       public static void main(String[] args) {           final String TEXT = "This \t is a \t\t\t String";           // Khoảng trắng xuất hiện 1 hoặc nhiều lần.         String regex = "\\s+";           Pattern pattern = Pattern.compile(regex);           Matcher matcher = pattern.matcher(TEXT);           int i = 0;         while (matcher.find()) {             System.out.print("start" + i + " = " + matcher.start());             System.out.print(" end" + i + " = " + matcher.end());             System.out.println(" group" + i + " = " + matcher.group());             i++;         }       }  } |

Kết quả chạy ví dụ:



Method **Matcher.lookingAt()**

* **MatcherLookingAt.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | package org.o7planning.tutorial.regex;    import java.util.regex.Matcher;  import java.util.regex.Pattern;    public class MatcherLookingAt {       public static void main(String[] args) {         String country1 = "iran";         String country2 = "Iraq";           // Bắt đầu bởi I tiếp theo là ký tự bất kỳ.         // Tiếp theo là ký tự a hoặc e.         String regex = "^I.[ae]";           Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);           Matcher matcher = pattern.matcher(country1);           // lookingAt() tìm kiếm khớp phần đầu.         System.out.println("lookingAt = " + matcher.lookingAt());         // Trong khi matches() phải khớp toàn bộ         System.out.println("matches = " + matcher.matches());           // Reset matcher với text mới, country2         matcher.reset(country2);           System.out.println("lookingAt = " + matcher.lookingAt());         System.out.println("matches = " + matcher.matches());     }  } |

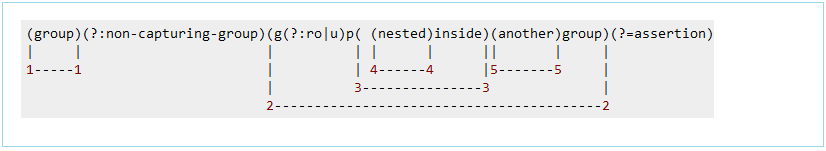
6- Nhóm (Group) [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

Một biểu thức chính quy bạn có thể tách ra thành các nhóm (group):

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | // Một biểu thức chính quy  String regex = "\\s+=\\d+";    // Viết dưới dạng group, bởi dấu ()  String regex2 = "(\\s+)(=)(\\d+)";    // Một cách khác.  String regex3 = "(\\s+)(=\\d+)"; |

Các group có thể lồng nhau, và như vậy cần một quy tắc đánh chỉ số các group.  Toàn bộ pattern được định nghĩa là group số 0. Còn lại được mô tả giống hình minh họa dưới đây:



*Chú ý: Sử dụng (?:pattern) để thông báo với Java không xem đây là một group (None-capturing group)*

Từ Java 7, bạn có thể xác định một group có tên (?<name>pattern), Và bạn có thể truy cập các nội dung khớp với Matcher.group (String name). Điều này làm Regex dài hơn, nhưng mã này là có ý nghĩa hơn, dễ hơn.  
  
Nhóm bắt theo tên cũng có thể được truy cập thông qua Matcher.group (int group) với các đề án đánh số tương tự.  
  
Nội bộ, Java chỉ lập bản đồ từ tên đến số nhóm. Do đó, bạn không thể sử dụng cùng tên để bắt 2 nhóm khác nhau.

-

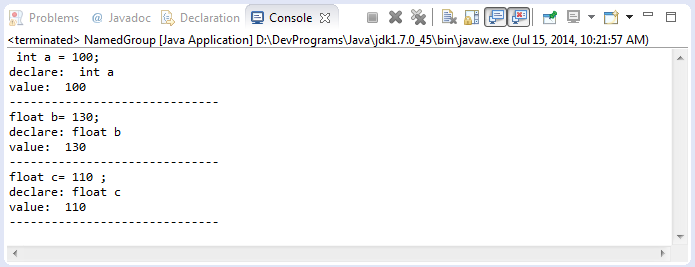
Hãy xem một ví dụ sử dụng đánh tên cho nhóm (group) (Java >=7)

* **NamedGroup.java**

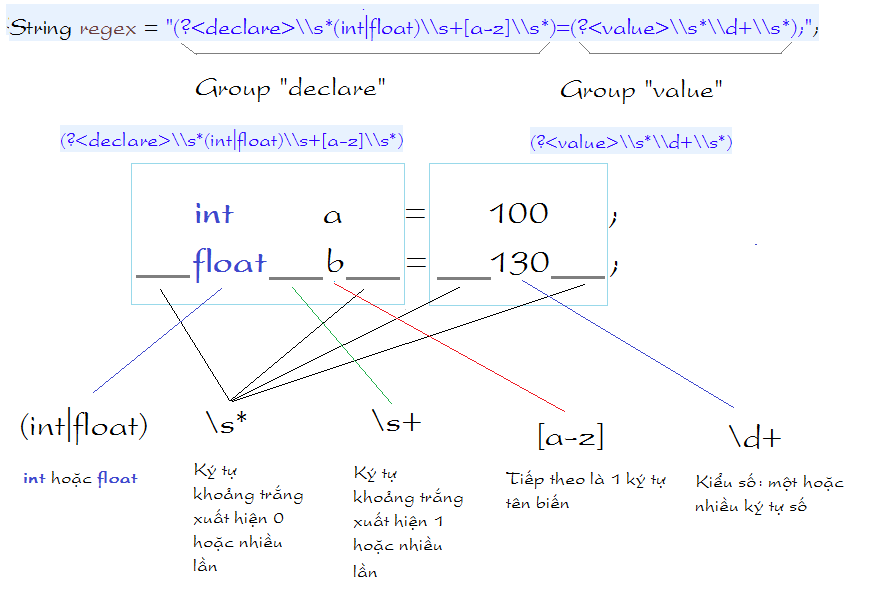
[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | package org.o7planning.tutorial.regex;    import java.util.regex.Matcher;  import java.util.regex.Pattern;    public class NamedGroup {       public static void main(String[] args) {             final String TEXT = " int a = 100;float b= 130;float c= 110 ; ";           // Sử dụng (?<groupName>pattern) để định nghĩa một Group có tên: groupName         // Định nghĩa group có tên declare: sử dụng (?<declare> ...)         // Và một group có tên value: sử dụng: (?<value> ..)         String regex = "(?<declare>\\s\*(int|float)\\s+[a-z]\\s\*)=(?<value>\\s\*\\d+\\s\*);";           Pattern pattern = Pattern.compile(regex);           Matcher matcher = pattern.matcher(TEXT);           while (matcher.find()) {             String group = matcher.group();             System.out.println(group);             System.out.println("declare: " + matcher.group("declare"));             System.out.println("value: " + matcher.group("value"));             System.out.println("------------------------------");         }     }  } |

Kết quả chạy ví dụ:



Để dễ hiểu bạn có thể xem hình minh họa dưới đây:

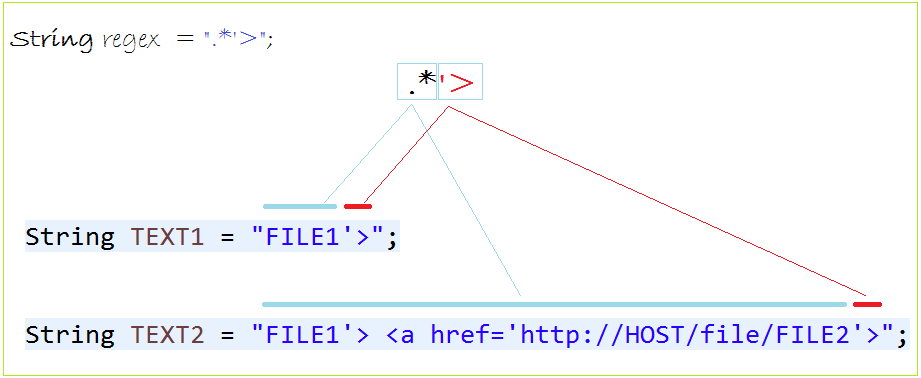


7- Sử dụng Pattern, Matcher, Group và \*? [▴](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex" \l "top)

Trong một số tình huống \*? rất quan trọng, hãy xem một ví dụ sau:

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | // Đây là một regex  // Bắt gặp ký tự bất kỳ 0 hoặc nhiều lần,  // sau đó tới ký tự ' và tiếp theo là >  String regex = ".\*'>";    // Đoạn TEXT1 sau đây có vẻ hợp với regex nói trên.  String TEXT1 = "FILE1'>";    // Đoạn TEXT2 sau cũng hợp với regex nói trên.  String TEXT2 = "FILE1'> <a href='[http://HOST/file/FILE2](http://host/file/FILE2)'>"; |



\*? sẽ tìm ra một phù hợp nhỏ nhất. Chúng ta xem ví dụ sau:

* **NamedGroup2.java**

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | package org.o7planning.tutorial.regex;    import java.util.regex.Matcher;  import java.util.regex.Pattern;    public class NamedGroup2 {      public static void main(String[] args) {        String TEXT = "<a href='[http://HOST/file/FILE1](http://host/file/FILE1)'>File 1</a>"                + "<a href='[http://HOST/file/FILE2](http://host/file/FILE2)'>File 2</a>";          // Java >= 7.        // Định nghĩa một group có tên fileName        // \*? ==> Nó sẽ tìm một phù hợp nhỏ nhất.        String regex = "/file/(?<fileName>.\*?)'>";          Pattern pattern = Pattern.compile(regex);        Matcher matcher = pattern.matcher(TEXT);          while (matcher.find()) {            System.out.println("File Name = " + matcher.group("fileName"));        }    }    } |

# Best Practices For Memory Optimization on Android

For those tracking [the evolution of Android](http://blog.hsc.com/android/technology-trends/embedded-systems-history/), it is evident that the future of the Android based ecosystem goes far beyond just phones and tablets. The OS is already making its way into a host of other [smart devices](http://blog.hsc.com/android/android-devices-at-home/embedded-android-devices-quick-look-todays-smart-technology-powered-android-os/), like [Google Glass](http://blog.hsc.com/android/technology-trends/google-glass-development/) for example, in a movement toward what’s being called “[the internet of things](http://blog.hsc.com/android/technology-trends/internet-of-things/)” or IoT. Developing a new OEM product based on Android as an embedded OS makes a lot of sense compared to say, only using Linux as we have [covered before](http://blog.hsc.com/android/technology-trends/android-vs-linux/). However, getting Android to actually work effectively on diverse platforms is quite challenging. While phones and tablets are getting very powerful (with quad core processors and 2+GB RAM having become the de facto standard) this is certainly not the case with many other IoT devices where due to cost margins, the need of the day is still lower powered CPUs and lesser RAM (as RAM is an expensive part of any device BOM). While there are many mechanisms to reduce Android footprint and reduce memory overhead (such as headless Android mode, low memory Android configurations, etc.) ensuring that the application code also effectively uses available memory is important. This article covers best practices for memory usage.

## Android & Ram

There are two kinds of memories when it comes to Android: Clean RAM and Dirty RAM.

### Clean RAM

Unlike PCs, Android does not offer swap space for memory, however it does use paging and memory-mapping. Any files or resources which are present on the disk, such as code, are kept in mmap’ed pages. Android knows these pages can be recovered from the disk, so they can be paged out if the system needs memory somewhere else.

### Dirty RAM

Dirty RAM is the memory that cannot be paged out. It can be expensive, especially when running in a background process. Most of the memory in a running application is dirty memory and this is the one you should watch out for.

In order to optimize the memory usage, Android tries to share some framework resources or common classes in memory across processes. So whenever a device boots up, a process called zygote loads the common framework code. Every new application process is then forked from the zygote process so it is able to access all the shared RAM pages loaded by it. While investigating an application’s RAM usage, it is important to keep shared memory usage in mind since we should only be looking at the private dirty memory that is being used by our application. This is reflected by USS (unique set size) and PSS (proportional set size) in ‘meminfo.’

Another important thing to keep in mind when investigating opportunities for memory optimization is that Android divides the application processes based on **running vs cached processes**. A running process is the foremost application running on the device or an application with a service running actively in the background. All other launched applications will go into the list of cached processes to allow for easier and faster switching between applications. For example, if an application is launched and then the user presses the ‘Home’ button, then that application’s process will be added in the list of cached processes, that is if it does not have a running service. Be aware that the system will kill one or more cached processes if it needs more memory for any running process. Cached processes can be killed in LRU (least recently used) order. However there are some other options also, like killing whichever cached process will give the maximum memory gain for the system.

Every application on Android has a maximum **heap size limit** (varies for each device). You can check the getMemoryClass() API of ActivityManager service, it will tell you the maximum heap size available for any application on a device. Most devices running Android 2.3 or later will return this size as 24MB or higher. For example, the size on a Galaxy S3 is 64MB, whereas on a Nexus5 device, it is 192MB. Android will always start an application process with an average heap size and will then grow it up to the maximum limit on that device for an app. If an application reaches the maximum heap size and needs to allocate more memory, the system will throw an OutOfMemoryError.

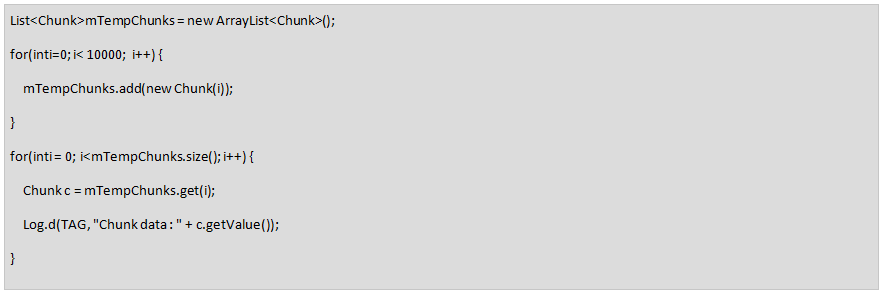
## Memory Optimization, Best Practices For Android

So what can you do to keep your system from running out of memory? Read on for some general guidelines for improving the memory usage and overall performance of Android.

### Avoid Creating Unnecessary Objects

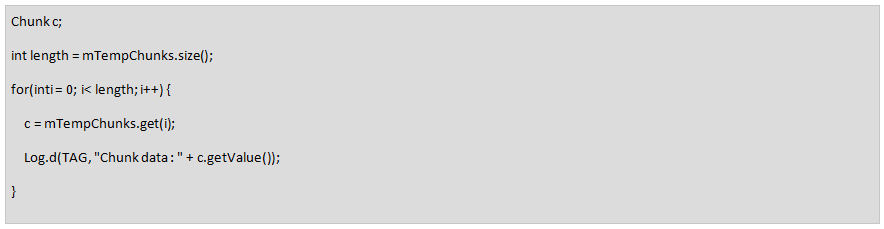
The basic rule of thumb here is that garbage collection is not free. The more objects an application allocates, the more frequently garbage collector will be forced to run – which eats up resources needed to boost user experience and responsiveness. Temporary objects can also hurt. A large number of small allocations can also cause heap fragmentation.

**For Example:​**



In the second loop of the code snippet above, we are creating a new chunk object for each iteration of the loop. So it will essentially create 10,000 objects of type ‘Chunk’ and occupy a lot of memory. Imagine, if we have just missed the routine GC cycle before creating these objects, then these objects would lay around until next GC.

The same code can be written like the version below, instead:



Just one object!

**Another Example:**

Try to reuse the same object when passing through functions, like below:



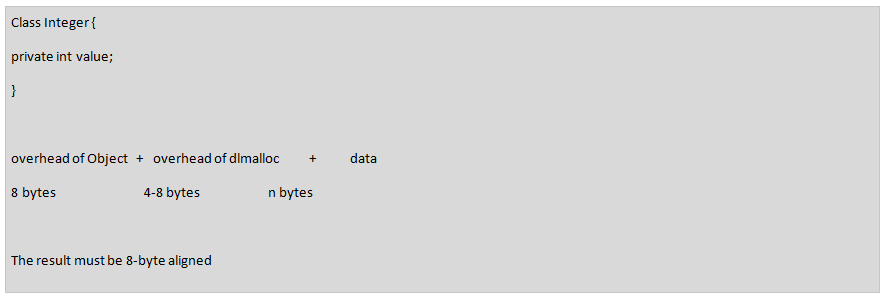
Notice how the string is appended to same StringBuilder object directly without creating any short term temporary objects for String and StringBuilder.

### Be Aware Of Memory Overhead Of The Language

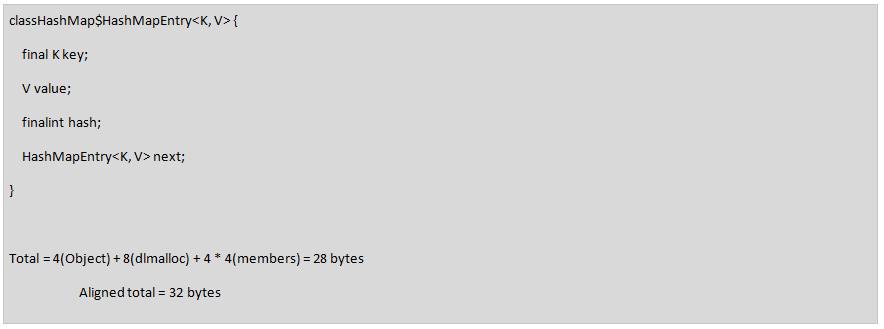
It helps to know the cost and overhead of language constructs we are using.

**For example:**

An Object with just one int variable takes 16 bytes at minimum in Android:



HashMap:



So, every entry in a HashMap would occupy 32 bytes.​

### Objects Vs. Primitive Types

As mentioned above, since an “Integer” boxed object occupies 4 times as much memory as primitive “int,” we should always try to use primitive types where we can. Similarly, a Boolean boxed object occupies much more memory than primitive boolean type. In the example below, the API call returns a primitive “int” value but we have assigned this value to an Integer object. This assignment will perform an autoboxing operation from int to Integer object. For a single call, this may not matter much. However, if we are using it frequently, for example in an inner loop, it might occupy a lot of memory unnecessarily.​



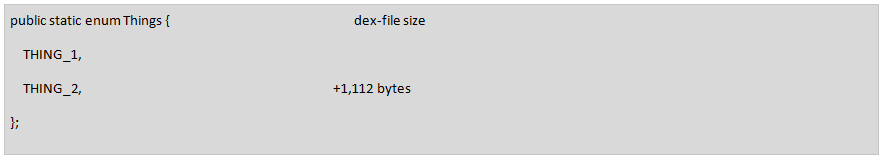
Integer (16 bytes)                 vs                 int(4 bytes)

Boolean(16 bytes)                vs                 boolean(4 bytes)

vs                 bit-field(1 bit)                 //even better!

### Enums Vs. Ints

Plain and simple – always avoid using enums on Android. Instead, use “static final” variables for constants. Enums usually require more than twice as much memory as static constants.

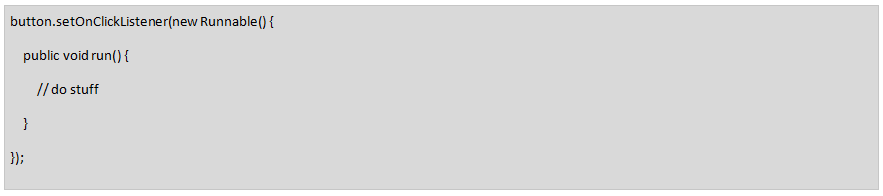


vs.



### Avoid Unnecessary Classes/Inner Classes

Every class in Java, including anonymous inner classes which create an object and writes accessor methods internally, uses about 500 bytes of code.



So, these kinds of listeners should be unregistered as soon as they are not needed.

### Hidden Cost Of Abstractions

In general, writing code with multiple layers of abstraction is considered good programming practice for object-oriented languages. However, the more code that is written, the more execution time and memory it is going to take. So, try not to overdo the layers. Only use abstractions where they provide a significant benefit. For example, in cases where writing a library to be used by other applications, it makes sense to use abstractions to expose only certain areas of functionality.

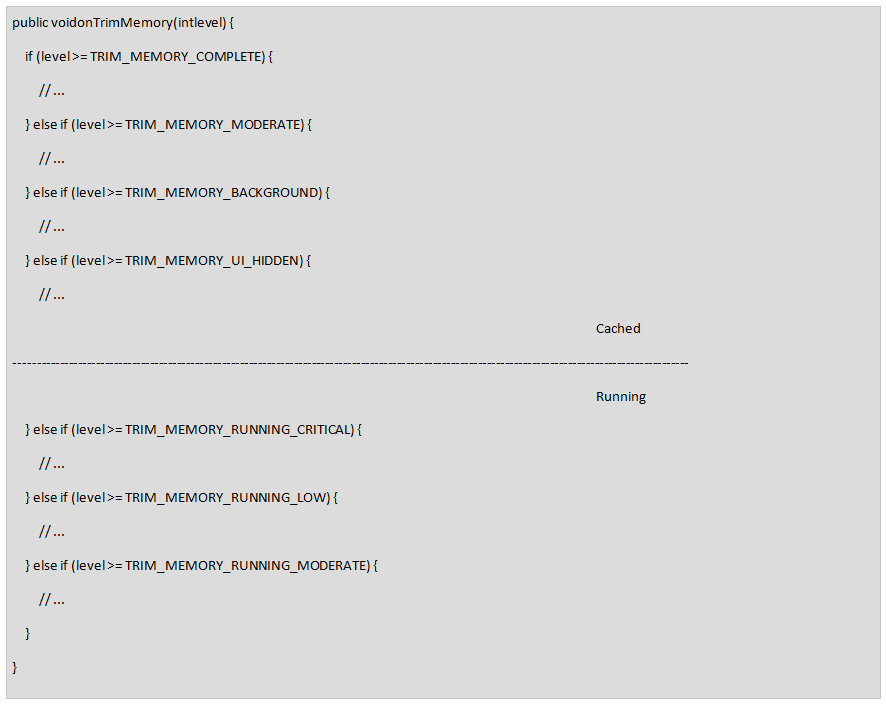
### Beware of Services

Services are useful for running operations in the background, but they are very expensive. You should never keep a service running unless absolutely required. The best way to automatically manage service lifecycle is to use an IntentService, which will finish itself after its work is done. For other services, it’s the application developer’s responsibility to make sure that stopService or stopSelf is being called after work is done.

### Release Memory When User Interface Becomes Hidden

When the user navigates to a different activity, release the resources associated with that activity in onPause and onStop callbacks. These resources are generally a network or database connection, a broadcast receiver, etc.

If the user navigates to a different application and all the UI components of the app are hidden, the app receives onTrimMemory() callback in all activities if Android system needs to kill any cached process to reclaim some memory for a running process. Listen for the TRIM\_MEMORY\_UI\_HIDDEN level and release the UI resources here. For example, textviews, imageviews, etc.



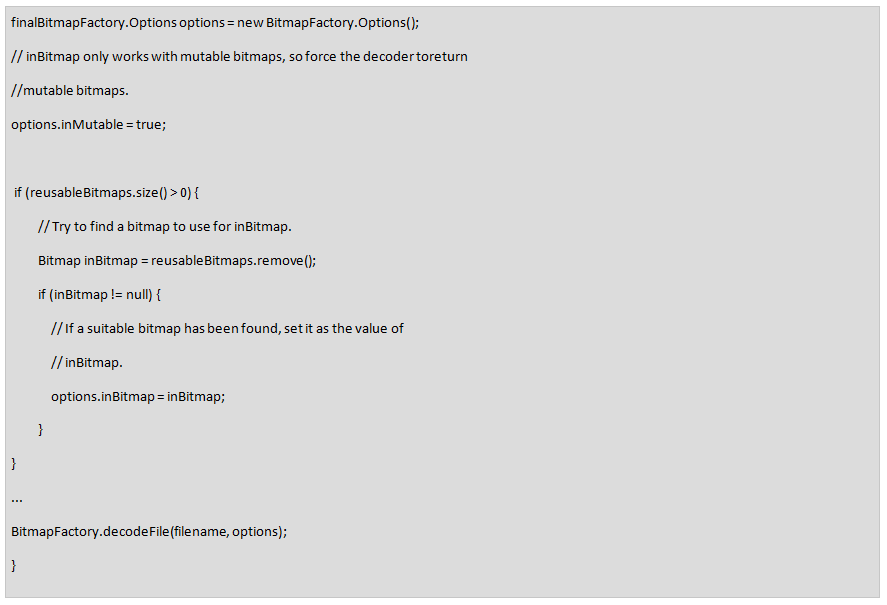
### Optimize Bitmaps Memory Usage

Bitmaps are often the largest RAM user in an application. A bitmap loaded in memory takes much more RAM than the size of the image we see on filesystem because:

bitmap size = width \* height \* depth (usually 4 bytes)

Keeping it in mind, bitmap should be loaded in RAM only at the size and resolution of the current device’s screen. So, we should scale it down if the original bitmap is at higher resolution. On Android 2.3.3 and lower, the backing pixel data for a bitmap was stored in native memory, and there was a need to write finalizers in Java code to free this native memory allocation. Therefore, it used to take more than one GC cycle to free bitmap memory. Hence it was recommended to use recycle on bitmaps after using them to free the memory as soon as possible.

As of Android 3.0 (API level 11) however, the pixel data is stored on the Dalvik heap along with the associated bitmap. So there is no need to call recycle(). But it is still useful to optimize the large amount of memory used by bitmap, and we should try to reuse bitmaps whenever possible. API level 11 introduces the BitmapFactory.Options.inBitmap field. If this option is set, decode methods of BitmapFactory that take the Options object will attempt to reuse an existing bitmap when loading content.



### Use Optimized Datacontainers

Android has provided a few optimized data containers in SDK and support libraries, such as SparseArray, SparseBooleanArray, and ArrayMap. These containers can be a replacement for Hashmap where the keys are of primitive type like int, Boolean, and so on. Since HashMap needs an Integer object for storing ints, it occupies a lot more memory than is actually needed, especially if we have large number of entries in our map. ArrayMap also consumes less memory, however it is slower in access than HashMap, so it should be only used when working with smaller number of elements, like < 100.

Here are some examples of optimized data containers which we can use as replacement for their HashMap equivalents:

|  |  |
| --- | --- |
| HashMap | Array Class |
| ​<Integer, Object> | SparseArray |
| ​<Integer, Boolean> | SparseBooleanArray |
| ​<Integer, Integer> | SparseIntArray |
| ​<Integer, Long> | SparseLongArray |
| ​<Long, Object> | LongSparseArray​ |

Use raw arrays, like int[], in performance-critical sections of the code or where we are working with hundreds of thousands of elements at a time, if possible.

### Proguard And Zipalign

The ProGuard tool shrinks, optimizes, and obfuscates the code by removing unused code and renaming classes, fields and methods with semantically obscure names. ProGuard can make the code more compact, requiring fewer RAM pages to be mapped. But you should be aware of how ProGuard works before using it in the application. For example, by default, ProGuard will strip out native JNI functions, dynamically loaded classes or methods, and code which is part of some library internally referenced by another library in the project. So, it is important to configure ProGuard config file to add rules for keeping all the required classes and methods in the project.

While preparing a release build of the application, it is always important to run the ZipAlign tool on the APK to have it re-aligned. This is necessary in order to maximize our static code and resources to be mmapped by Android. Eclipse probably already does that automatically, however we should take care if we are building the APKs on our own using Ant.

To recap, some **general performance tips for memory optimization**:

* Try to avoid static variables or objects as much as possible if they are not final constants. Static variables pose the threat of having references in other classes, which we might forget about and thus cause a memory leak.
* Prefer static methods over virtual methods where any of the member fields of object are not accessed. The static invocations are faster because they save dalvik look up of the method.
* Avoid internal getters and setters. Direct field access is much faster in Android than virtual method lookup. If an app is not exposing APIs as libraries, it should avoid using accessors.
* A common mistake is to save “Context” objects everywhere in the application. If the developer forgets to free even one reference of a context, it increases the chance of a whole activity leak.
* Match the calls of registration and un-registration of the listeners, receivers in corresponding pairs in activity lifecycle. If a broadcast receiver was registered in onStart method, then it should be unregistered in onStop method only. The same goes for onCreate-onDestroy and onResume-onPause.
* In Eclipse or Android Studio, ADT comes with lint analysis tool. It provides tips and tricks for optimizing the application code at the time of compilation. Always pay attention to these tips and warnings and try to incorporate them if possible.
* For passing information between components of same application, avoid using IPC mechanisms like Intents and ContentProviders if possible. Work with handlers, listener interfaces instead.

## Tools For Measuring Android Memory Usage

To analyze an application’s memory usage on Android, there are several memory profiling tools available. The Android SDK provides two main ways of profiling the memory usage of an app: the **Allocation Tracker** tab in DDMS, and **heap dumps.** The Allocation Tracker is useful when we want to get a sense of what kinds of allocations are happening over a given time period, but it doesn’t provide any information about the overall state of the application’s heap.

**To collect a heap dump:**

* Run the app
* Select the app in DDMS
* Press “Dump HPROF File” button
* Save file
* Run hprof-convhprof-conv heap-original.hprof heap-converted.hprof

Note – DDMS version integrated into Eclipse does hprof conversion automatically, so this step is not required.

To analyze a heap dump, we can use standard tools like **Eclipse Memory Analyzer(MAT)** or **jhat**.

When analyzing the heap dump, **look for memory leaks caused by:**

* Long-lived references to an Activity, Context, View, Drawable, and other objects that may hold a reference to the container Activity or Context.
* Non-static inner classes (such as a Runnable, which can hold the Activity instance).
* Large objects accumulating over time collect multiple heap dumps at different intervals and compare them. MAT Tool’s Leak Suspects Report is particularly useful in this area.

### 

### Meminfo And Procstats

To observe how an application is divided between different types of RAM, we can use the following adb command:

adb shell dumpsys meminfo <package\_name>

This command lists all the current allocations of an application, measured in kilobytes. The important details are the memory used by USS (Private Dirty + Private Clean) and PSS total. It will also show the number of activities currently running, number of view objects allocated and binder objects shared between processes.

Kitkat (4.4) introduced a new service called Procstats to help better understand the memory usage on a device. There is a UI screen found under the Developer Options menu so that you can look at the memory usage by all applications. To run Procstats from command line:

adb shell dumpsys procstats <package\_name>

(Do you find this article interesting? You may want to check out our [Embedded Android](http://hsc.com/Services/Product-Engineering-Services/Embedded-Android) pages to read more about what we do in this space.)