Professional Development Program Series, Vol. I, No. 1, 1-5, 2014

Java Apprentice Badge Report

Marko Viitanen[1]*

Abstract
As part of the Company Professional Development Program we are provided wasy to learn and demonstrate our development skills by earning badges. There are several badges, Java being one of them. This document describes my learnings and answers to the requirements for earning the Apprentice Java Badge.

Keywords
Java — Professional Developmen Program — Badges

[1] *FamilySearch, Salt Lake City*

Contents

Introduction

The Java Apprentice Badge is the first badge in the Java series. It covers intermediate concepts of Java programming. It is not a 101 course in programming, so many of the common language features are not covered. The requirements don't include installation,

High-level requirements[1] include:

- Object life cycle
- Exceptions
- Polymorphism
- Collections
- Using a library

Each apprentice is allowed to creatively demonstrate their knowledge on the required topics. I chose to write a report about it.

[1] See confluence for full description of the requirements at https://almtools.ldschurch.org/fhconfluence/display/Product/Core+Skills+-+Java+-+Apprentice .

1. Core Java

The first section deals with Java primitives, objects and their life cycle, and some JDK-provided classes for String manipulation and Collections. It includes writing applications to sort Strings. It also deals with Exceptions and Enums.

1.1 Object Life cycle

Describe the life cycle of an object instance in Java and how garbage collection works

Procedural programming preceded object-oriented programming. Basically, The code was written line by line, like a recipe. The computer would execute the lines in the order they appeared in the program. There were constructs to jump from one place to another, called gotos. But very quickly a program can become quite complicated, and specially the gotos would make it very hard to follow.

With many lines of code, the program becames difficult to maintain. We can split the program into several files and import the pieces when compiling. That helps humans to organize the data but also fulfills the need for the compiler to have all the pieces of the program available. It doesn't solve the problem of scope, though. In procedural programming the data is accessible to the entire program. There is no clear ownership of data.

Procedural programming also provided constructs called subroutines. They are blocks of code, collections of instructions. Subroutines have their own scope, but any data they need to access outside the subroutine is still exposed to the entire program.

Java is an object-oriented language. In object-oriented languages, instead of having data and subroutines, we deal with objects that have data and behavior. WIth object-oriented programming we can easily model the real world. For example we can have an object of a dog that has data (color, breed) and behavior (barks, runs, drools).

When developers write java programs, they write classes. A class is like a blueprint of an object, it defines the object. A

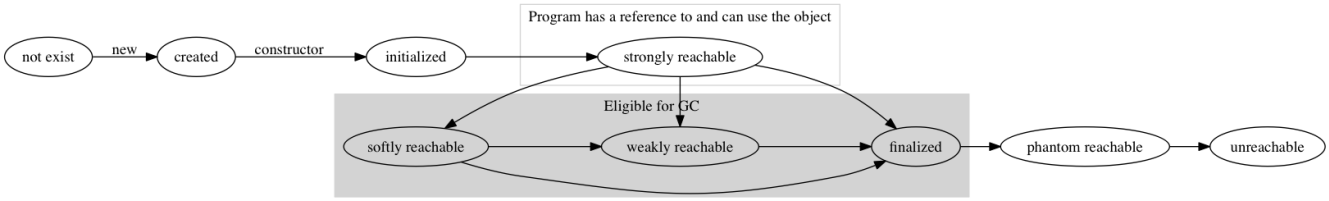

Figure 1. Object Life Cycle

class becomes an object when we instanciate it, we create an instance of it. Obects live when the program is executing, at runtime.

With objects it is easy to encapsulate behavior and data. We can restrict access to the data to only the members inside of a class. Nobody outside the class can access the data, if we don't want to (and we shouldn't want to.) We can define interfaces that provide indirect, controlled access to the data inside the class.

Organizing code becomes easier too, because each file can only have one public class per file. Since each class encapsulate one "thing", that has a well-defined interface that determines its behaviors, the code becomes very logical.

Everything in Java is made of another object. We call this inheritance. Java provides the mother of all objects, called `Object`, from which any other class must inherit.

Instantiating a class. We create an object from a class with the Java keyword `new`. Before instantiation, the object doesn't exist. After instantiation it exists. Classes have a special method called a constructor. After creating the class the Java Virtual machine (JVM) calls the object's constructor. If you don't provide a constructor for your class, the its parent (eventually the `Object` constructor is called. The constructor is a place where you coudl initialize the object or start resources.[5]

Strongly Referenced. When the constructor has been called, your program has a strong reference to it.[6] It means you can access the non-private methods and data on it. It is usable by your program.

```
1 Dog pepper = new Dog();
```

In the above example, "pepper" is the handle to your object, or a reference. It is a strong reference because you can use it to do things with the "pepper" object:

```
1 pepper.bark();
```

You can have several references to the same object:

```
1 // create an instance of Dog
  Dog pepper = new Dog();
3
  // pepperClone also points to the same object
5 pepperClone = pepper;
```

```
7 // set pepper's name to "pepper"
  pepper.setName("pepper");
9
  // returns "pepper"
11 pepperClone.getName();
```

In the above example we created an instance of a Dog and got back a reference to it called pepper. Then we made pepperClone also point to the same object. After that we set the name of pepper to "pepper". Because pepper and pepperClone point to the exact same object, when we ask pepperClone for its name, we get "pepper".

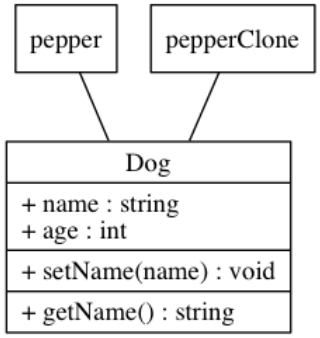

Figure 2. Object References

Other references. Once you let go of all the references to an object, it becomes eligible for garbage collection. The JVM still has a reference to the object (so it can manage it), but eventually, when it detects that momery needs to be cleaned up, it will finalize the object. JVM holds a weak or soft reference to the object.[6]

Garbage Collection. When the JVM determines that it needs to free memory, it will perform a garbage collection. The soft and weak references will be cleared before throwing an `OutOfMemoryException`.

Garbage collection is controlled by the JVM. There are tweaks you can do to suggest a certain behavior to the garbage collector, and you can even suggest that it will do garbage collection (generally not a good idea), but eventually the garbage collector will decide when to run.

The benefit of garbage colection is that the programmer doesn't need to think about finalizing objects. When they are not needed, tehy may be thrown into garbage automatically.

There are times when this thinking can get you into troublem though. If you don't release the references the objects will never be garbage collected.

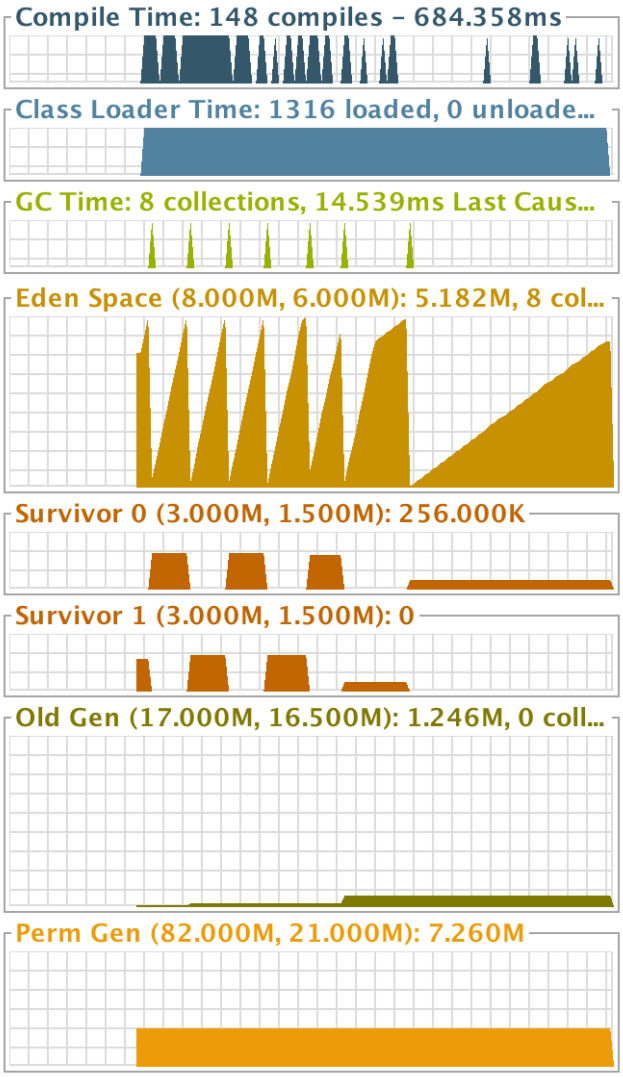

Figure 3. Garbage Collection

The above image shows garbage collection in action. I wrote a little program that creates objects and puts them in a collection. I used the visualvm tool provided in the Oracle JDK distribution[7]. After every 10000 objects I clear the collection. I wrote created a total of 306,480,000 objects, so I cleared the collection over 30,000 times. Garbage collection, though only kicked in 7 times. I had set my heap size to 25MB.

In the image you can also see the movement of objects from one generation to another.

1.2 Basic Data Types

Describe how the basic data types are represented in memory (boolean, int, long, String, array of ints, array of Objects, class with fields)

Java's primitive can be divided into two main groups: numeric primitives, and boolean. Numeric primitives consist of integral and floating point primitives.[1]

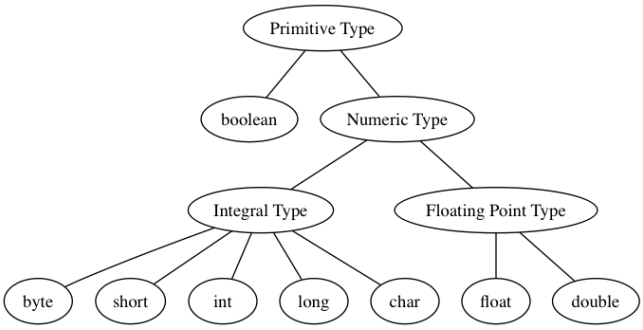

Figure 4. Primitive Classification

Java specification[1] defines sizes and ranges for primitive types. The integer types are clear cut, but the floating point types are not so simple. They follow the ANSI/IEEE Standard 754-1985, see http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3 for details. Those values are from my MacBookPro, printing out the max value for float and double. They might be different on a different architecture.

Also to note that although Java defines the primitive sizes, on different architectures might actually use different sizes. For example, although an int is defined as 32 bits, it might take 64 bits on a 64 bit computer. The primitive sizes defined in the Java Specification is how the programmer sees the types, not necessarily how they are stored.

Here is the list of primitives in Java[1]:

Type	Size (bits)	Range
byte	8	from -128 to 127, inclusive
short	16	from -32768 to 32767, inclusive
int	32	from -2147483648 to 2147483647, inclusive
long	64	from -9223372036854775808 to 9223372036854775807, inclusive
char	16	from '\u0000' to '\uffff' inclusive, that is, from 0 to 65535
float	32	from -3.4028235E38 to 3.4028235E38[3]
double	64	from -1.7976931348623157E308 to 1.7976931348623157E308[3]
boolean	1?[4]	true or false

String

array of ints

array of Objects

class with fields Reference to Figure 5.

[3] Java float and double follow the ANSI/IEEE Standard 754-1985, see http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3

[4] boolean size is not defined in the specification.

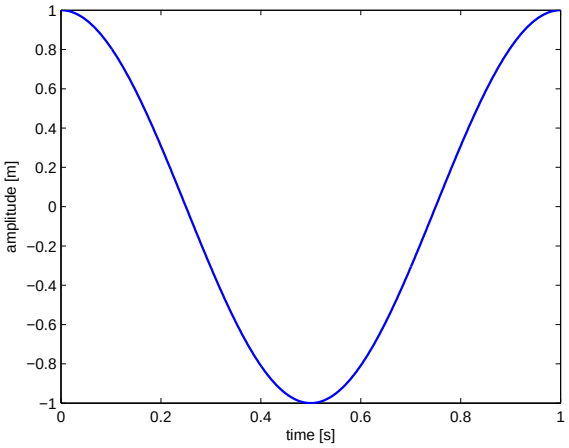

Figure 5. In-text Picture

2. Results and Discussion

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

2.1 Subsection

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Hello you

2.1.1 Subsubsection

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Word Definition

Concept Explanation

Idea Text

2.1.2 Subsubsection

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

- First item in a list
- Second item in a list
- Third item in a list

2.1.3 Subsubsection

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

2.2 Subsection

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.

Curabitur tellus magna, porttitor a, commodo a, commodo in, tortor. Donec interdum. Praesent scelerisque. Maecenas

posuere sodales odio. Vivamus metus lacus, varius quis, imperdiet quis, rhoncus a, turpis. Etiam ligula arcu, elementum a, venenatis quis, sollicitudin sed, metus. Donec nunc pede, tincidunt in, venenatis vitae, faucibus vel, nibh. Pellentesque wisi. Nullam malesuada. Morbi ut tellus ut pede tincidunt porta. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam congue neque id dolor.

Donec et nisl at wisi luctus bibendum. Nam interdum tellus ac libero. Sed sem justo, laoreet vitae, fringilla at, adipiscing ut, nibh. Maecenas non sem quis tortor eleifend fermentum. Etiam id tortor ac mauris porta vulputate. Integer porta neque vitae massa. Maecenas tempus libero a libero posuere dictum. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aenean quis mauris sed elit commodo placerat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Vivamus rhoncus tincidunt libero. Etiam elementum pretium justo. Vivamus est. Morbi a tellus eget pede tristique commodo. Nulla nisl. Vestibulum sed nisl eu sapien cursus rutrum.

Nulla non mauris vitae wisi posuere convallis. Sed eu nulla nec eros scelerisque pharetra. Nullam varius. Etiam dignissim elementum metus. Vestibulum faucibus, metus sit amet mattis rhoncus, sapien dui laoreet odio, nec ultricies nibh augue a enim. Fusce in ligula. Quisque at magna et nulla commodo consequat. Proin accumsan imperdiet sem. Nunc porta. Donec feugiat mi at justo. Phasellus facilisis ipsum quis ante. In ac elit eget ipsum pharetra faucibus. Maecenas viverra nulla in massa.

Nulla ac nisl. Nullam urna nulla, ullamcorper in, interdum sit amet, gravida ut, risus. Aenean ac enim. In luctus. Phasellus eu quam vitae turpis viverra pellentesque. Duis feugiat felis ut enim. Phasellus pharetra, sem id porttitor sodales, magna nunc aliquet nibh, nec blandit nisl mauris at pede. Suspendisse risus risus, lobortis eget, semper at, imperdiet sit amet, quam. Quisque scelerisque dapibus nibh. Nam enim. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nunc ut metus. Ut metus justo, auctor at, ultrices eu, sagittis ut, purus. Aliquam aliquam.

Etiam pede massa, dapibus vitae, rhoncus in, placerat posuere, odio. Vestibulum luctus commodo lacus. Morbi lacus dui, tempor sed, euismod eget, condimentum at, tortor. Phasellus aliquet odio ac lacus tempor faucibus. Praesent sed sem. Praesent iaculis. Cras rhoncus tellus sed justo ullamcorper sagittis. Donec quis orci. Sed ut tortor quis tellus euismod tincidunt. Suspendisse congue nisl eu elit. Aliquam tortor diam, tempus id, tristique eget, sodales vel, nulla. Praesent tellus mi, condimentum sed, viverra at, consectetuer quis, lectus. In auctor vehicula orci. Sed pede sapien, euismod in, suscipit in, pharetra placerat, metus. Vivamus commodo dui non odio. Donec et felis.

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

Donec et nisl id sapien blandit mattis. Aenean dictum odio sit amet risus. Morbi purus. Nulla a est sit amet purus venenatis iaculis. Vivamus viverra purus vel magna. Donec in justo sed odio malesuada dapibus. Nunc ultrices aliquam nunc. Vivamus facilisis pellentesque velit. Nulla nunc velit, vulputate dapibus, vulputate id, mattis ac, justo. Nam mattis elit dapibus purus. Quisque enim risus, congue non, elementum ut, mattis quis, sem. Quisque elit.

Acknowledgments

So long and thanks for all the fish [**?**].

Core Java

Write an application to find out how many total characters can be held in a list of strings before you run out of memory Compare and contrast StringBuffer and StringBuilder and when to use each Compare/contrast use of ArrayList / LinkedList / HashMap / HashSet / TreeSet Write an application to read a file with 10k lines of text, and output another file with the lines in sorted order (sample file) Write an application to read a file with 10k lines of text, and output another file with the lines in reverse sorted order Write code to show exception handling including examples of checked, unchecked, and Error exceptions Write your own enum type. Describe when you would use it. Working with Methods, Encapsulation and Inheritance Show how to use a common piece of logic from two different classes, in three different ways: 1) by composition, 2) by inheritance, and 3) by static method calls, discuss the tradeoffs for example: two different classes that write a message to a file, one in XML, one in line-oriented text, but both need to reuse logic to open the file in the same way Create and overload constructors – Create a class that has 4 fields and construct the class with variations of one required field and the others are optional. Use constructor chaining as an example. Apply encapsulation principles to a class – Show an example of good encapsulation. Show a bad example of encapsulation and explain why. Additionally explain access modifiers and how they can be used as part of the class encapsulation. Determine the effect upon object references and primitive values when they are passed into methods that change the values – Create a method 3 parameters, one is parameter is pass by value, one is passed by reference and one with the keyword final. Explain each and what the effects in side the method that changes each one. Write code to show how access modifiers work: private, protected, and public, talk about why you would use each of these. Write code to show how virtual method invocation lets one implementation be swapped for another. Write code that uses the instanceof operator and show how casting works. Show how to override

a method in a subclass, talk about plusses and minuses in doing so. Show how to overload constructors and methods, talk about plusses and minuses in doing so. Library Write an application that uses the slf4j logging library directly (can also choose log4j if you want) Do the following: configure the logging using an accepted department log statement format (see Application Logging) log at different logging levels (error, warn, info, debug), to see the effect of the default logging level setting turn on DEBUG in the logging config to show DEBUG output configure logging to go to both the console and a log file

References

[1] Gosling, James, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. *The Java® Virtual Machine Specification.* The Java® Virtual Machine Specification. Oracle America, Inc, 28 Feb. 2013. Web. 12 Aug. 2014. <https://docs.oracle.com/javase/specs/jvms/se7/html/>.

[2] Lindholm, Tim, Frank Yellin, Gilad Bracha, and Alex Buckley. *The Java® Virtual Machine Specification.* The Java® Virtual Machine Specification. Oracle America, Inc., 28 Feb. 2013. Web. 12 Dec. 2014. <https://docs.oracle.com/javase/specs/jvms/se7/html/>.

[3] *Java Platform SE 7.* Java Platform SE 7. Oracle, n.d. Web. 12 Dec. 2014. <http://docs.oracle.com/javase/7/docs/api/>.

[4] *Trail: Learning the Java Language.* The Java™ Tutorials. Oracle, n.d. Web. 12 Dec. 2014. <https://docs.oracle.com/javase/tutorial/java/index.html>.

[5] Nicholas, Ethan. *Understanding Weak References.* Understanding Weak References. Java.net, 4 May 2006. Web. 13 Dec. 2014. <https

[6] *Package java.lang.ref.* Java.lang.ref (Java Platform SE 7). Oracle, n.d. Web. 13 Dec. 2014. <http://docs.oracle.com/javase/7/docs/api/java/lang/ref/package-summary.html>.

[7] *Java Garbage Collection Basics.* Java Garbage Collection Basics. Oracle, n.d. Web. 13 Dec. 2014. <http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>.