Objectives
At the completion of this lab, students will be able to
- Write, compile, and debug Java code using Eclipse
- Identify problems with reference aliases
- Improve encapsulation of class designs that might have aliases problems

Pre-Lab Activities
Before you come to the lab, you must do these activities.
- Review your Java knowledge by reading LC Appendix, in particular the sections that cover objects and references
- Read section H3.4.5, H1.5, H1.6
- Bookmark the Java 1.4 API documentation in your favorite browser and look up the class documentation for String class. Is there a method that allows changing the content of a String?

Lab Activities
We will be using Eclipse for the lab exercises and we strongly encourage you to give it a try on your own computer. It is available for free at www.eclipse.org and it is highly extensible with support for multiple languages, including C, C++, Java, C# and others. If you want to explore Eclipse before the lab, check the tutorial available at http://www.3plus4software.de/eclipse/index_en.html. Access the Eclipse installation instructions on the course web site.

A. Tour of Eclipse
- Find Eclipse on your computer and launch it.
- Follow the “Basics of Java” development tour of Eclipse, (from the above URL), being shown in the lab. The major sections will be shown in class.
  - How to create a new project
  - How to create a class/package
  - How to browse code, compile, and debug
  - Location of files in Eclipse
  - How to use the type completion
  - How to use the debugger: breakpoints, stepping into, out of, etc.

B. Aliases & More Eclipse
1. Create a new project in Eclipse, a new package (call it edu.vt.cs.cs1706.yourpid.lab1) and add the following class to it.
```java
class Message {
    private String s;
    public Message (String s) {
        this.s = s;
    }
    public void setMessage(String s) {
        this.s = s;
    }
    public String getMessage() {
        return this.s;
    }
}
```

2. Create a new class that will contain your main method and instantiate an object of the Message class.
```java
class Lab1_2 {
    public static void main(String[] args) {
        Message msg1 = new Message("hello");
        Message msg2 = msg1;

        // here in the main method:
        // how many objects do we have?
        // how many references do we have?
    }
}
```

3. How many objects have I explicitly created in the main program? Where do msg1 and msg2 point to? Add the following two lines to the end of the main method and run this program. What is wrong with the following code?
```java
System.out.println(msg1.getMessage());
System.out.println(msg2.getMessage());
```

Add the following three lines to the end of the main method and run this program again.
```java
msg1.setMessage("hola");
System.out.println(msg1.getMessage());
System.out.println(msg2.getMessage());
```

3. How can you explain the behavior in the previous program?
4. A bit more of Eclipse. Have an opportunity to do some refactoring. Refactoring is "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior" [Martin Fowler, “Refactoring”, Addison-Wesley]. Observe the refactoring demo.

C. Argument Passing, Values and References
1. Add a method to your Message class to do the following. The goal of this method is to set an instance of Message that is received as an argument to be the same as the object receiving the call.

   ```java
   public void setToSame(Message m) {
       m = this;
   }
   ```

2. Given what you now know about references, how would you test this method? How do you know it works correctly? We will do test classes in the next part of this lab; for now, write a main program that calls this method and either print something out or do a condition that verifies that this works as expected.

   Discussion: What is wrong with this code?

D. Identity vs. Equality
For languages that support references, you have to deal with two similar but slightly different concepts: identity and equality. The concepts are best defined informally as following:

Identity – two object references have the same identity if they both point to the same object. This is the situation when we have aliases; two object references have the same identity. In the example below, msg1 and msg2 have the same identity; both references point to the same location, there is a single object.

   ```java
   Message msg1 = new Message("hi");
   Message msg2 = msg1;
   ```

Equality – two objects are equal if their internal values are equal. In the example below, both objects msg1 and msg2 have the same value inside but are different instances of the Message class.

   ```java
   Message msg1 = new Message("hi");
   Message msg1 = new Message("hi");
   ```

Try the following:

1. Check the methods in the String class (using your bookmark to your Java API). Which method compares two strings to see if their contents are the same?
2. What is printed below? And why?
String msg1 = "hi";
String msg2 = msg1;

System.out.println(msg1 == msg2);
System.out.println(msg1.equals(msg2));

3. What is printed below? And why?

String msg1 = "hi";
String msg2 = "hi";
String msg3 = "hasta la vista, baby";

System.out.println(msg1 == msg2);
System.out.println(msg1.equals(msg2));

System.out.println(msg1 == msg3);
System.out.println(msg1.equals(msg3));

**Word of caution:** In Java, the operator == tests for identity.

4. What is printed below? And why?

Message msg1 = new Message("hi");
Message msg1 = new Message("hi");

if (msg1 == msg2) {
    System.out.println("they are the same");
} else {
    System.out.println("they are not");
}