

## 1 Homework 2: Due Monday September 7, 2009

Since Eclipse will be the primary development environment for the course, your goal is to become comfortable and productive using Eclipse.

## 2 Objectives

- Learn how to use Eclipse.
- Practice using Eclipse.

## 3 Introduction

In this homework you will learn how to use Eclipse. Eclipse is an *Integrated Development Environment* (IDE) for developing Java software. Eclipse is widely used in industry because it makes most software development tasks easier. For example, Eclipse will allow you to edit your Java programs and provide immediate feedback on syntax errors. Eclipse can perform many mundane tasks for you. For example, it can insert Javadoc comments for methods. Since Eclipse is an industrial-strength software development tool, it can be intimidating for the beginner, however, it is well worth the effort to learn how to use Eclipse.

## 4 Getting Started With Eclipse

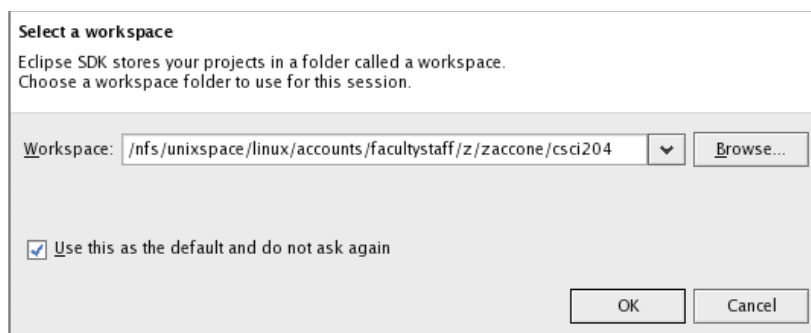
Begin by opening a terminal window and making a directory that you will use for this course.

```
mkdir csci479
```

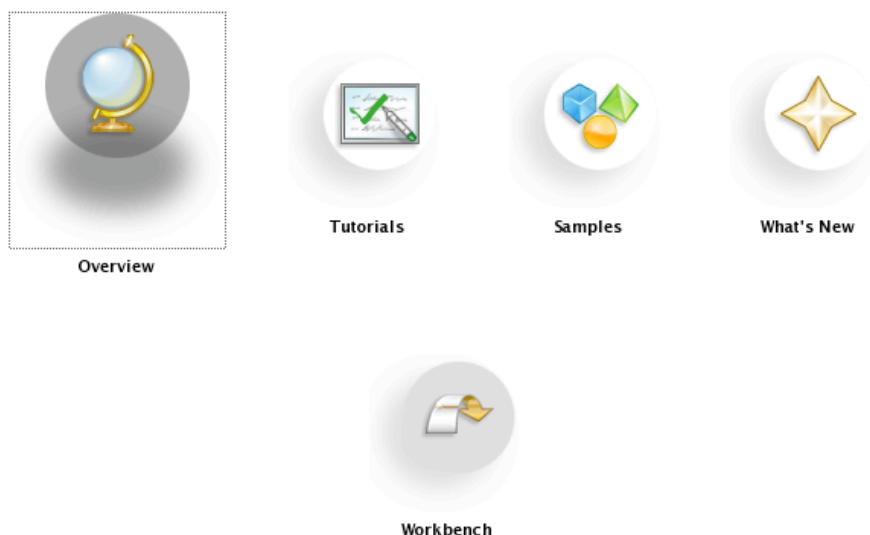
Then, start Eclipse by either typing

```
eclipse &
```

in your terminal window, or by selecting Eclipse from the Programming menu found within the Red Hat menu. Eclipse will present you with a dialog asking you for the location of your workspace. Indicate that it is your csci479 directory and check the box so it won't ask again.



When you start Eclipse you will be presented with the following welcome screen.



Click on the **Workbench** icon (the looping arrow) to begin using Eclipse.

## 4.1 Change Eclipse Settings

Follow the instructions in this section to make sure Eclipse is configured properly.

Open Eclipse preferences by selecting **Window** → **Preferences...**

- In **Java** → **Editor** → **Folding**, uncheck Header Comments and Imports. Enable folding should remain checked. Click the **Apply** button.
- In **Java** → **Editor** → **Save Actions**, check the box that says “Perform the selected actions on save”. Then check the box that says “Format source code.” Click the **Apply** button.
- In **Java** → **Code Style** → **Code Templates**, click the disclosure triangle next to Comments. Click on the word Files and then press the **Edit** button. Enter a comment similar to the one that follows, using your name.

```
/**
 * CSCI 479, Dan Hyde
 * ${date}, ${time}
 */
```

Click **OK**, then click the **Apply** button.

- In **General** → **Editors** → **Text Editors**, check the box that says “Show print margin” and then click the **Apply** button.
- In **Java** → **Installed JREs** → **Execution Environments**, select **JavaSE-1.6** then **OK** button.
- If needed, click the **OK** button to exit the preferences.

## 5 Create a Java Program

You are now ready to create your first Java program in Eclipse. It is an old computer science tradition to write a “Hello World” program as your first program. Since this is your first Eclipse program, that’s what we will do now. Follow the following steps closely.

- Select **File** → **New** → **Java Project**.
- Enter hw02 as the project name and press the **Finish** button.
- Select **File** → **New** → **Class**.
- Enter Hello as the name.
- Check the box that says to create a method stub for public static void main.
- Check the box that says to “Generate comments”.
- Click the **Finish** button.

You should be looking at a new class named Hello which has a `main()` method. There are comments too!

The comment

```
// TODO Auto-generated method stub
```

is there to remind you to fill in the method stub that Eclipse generated for you. You may remove it. Type the following text in its place

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

and save. To run the program, click the run button in the tool bar. It’s a green circle with a white arrow inside. (You may hover the mouse pointer over a button for a small window that explains the button.) You may get a **Run As** dialog. Select **Java Application** and press **OK**. Your output should appear in the console window at the bottom of the Eclipse screen.


### 5.1 Add Javadoc Comments

When Eclipse generated your `main()` method, it inserted Javadoc comments for you. Complete these comments now. (The content of your comments is not very important for this file.) Click on the Javadoc comment to edit. Click on the Javadoc tab at the bottom of the screen and then start editing the comment back above the `main()`. Eclipse will show you the *formatted* comment in the Javadoc pane. Edit the comment with the `@author` tag and add a description of your Hello class. Again, you should see the formatted comment in the Javadoc pane. Inside Javadoc comments you may use html tags such as `<b>` and `</b>` for bold text.

## 6 Scanner Class

In this section we will review how to use the Scanner class. While we learn about the Scanner class, we will also learn some features of Eclipse.

## 6.1 Create a New Class

As we saw before, you can create a new class by selecting **File** → **New** → **Class**. There is a toolbar icon for this too. It is a green circle with a white C in it . Click on this icon now to create a new class. Name your class `TestScanner`, ask for comments and a `main()` method, then press the **Finish** button.


Augment the Javadoc comment with the `@author` tag with a description of the class saying that it is an exercise in hw02 to practice using the Scanner class.

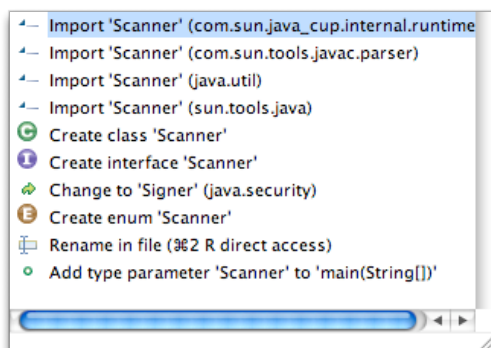
## 6.2 Ask Eclipse to Create an `import` Statements

A common problem is remembering the name of the `import` statement that you need to import a particular class. Why spend precious minutes searching through textbooks when you can have Eclipse do it for you?

Enter the following line into your `main()` method.

```
Scanner keyboard = new Scanner(System.in);
```

Eclipse interprets your program as you type. It immediately notices there is a problem because it doesn't know the `Scanner` class. It will underline both occurrences of the word `Scanner` in red and place a red rectangle with an X in the margin . Hover the mouse over this symbol or the underlined words, and Eclipse will tell you what's wrong. In this case it will tell you that `Scanner` is not a recognized type. Click on the red symbol in the margin, and it will offer suggestions on how to correct the problem.




Use the up and down arrow keys on your keyboard to move through this list. As each item is highlighted, Eclipse will show you what will happen if you select that item. In this case, you want to import `Scanner` from `java.util`. Select that item and press Enter. Eclipse will add the necessary `import` statement to your code.

Use Eclipse to enter the following code into `main()`. You can use the tab key to indent.

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter an int, double, word, and line: ");
int anInt = keyboard.nextInt();
double aDouble = keyboard.nextDouble();
String aWord = keyboard.next();
String line = keyboard.nextLine();
System.out.println(anInt + ":" + aDouble + ":" + aWord + ":" + line);
```

Run the program. Try typing in different values to see how the `Scanner` class works.

### 6.3 Eclipse Continuously Checks for Syntax Errors

Notice that Eclipse checks for syntax errors as you type. If it detects a syntax error, it will underline items in red and put a red rectangle with an X in the margin . Purposely make a few syntax errors such as spell “double” wrong to observe this behavior. Hover the mouse over this symbol or the words underlined in red, and Eclipse will tell you what’s wrong. If you don’t see the red X, save the file.

### 6.4 Eclipse Reformats the Program on a Save

Notice that when you save the file, Eclipse will indent and adjust the code to a standard style. It will also reformat `/* */` style comments. If you don’t want your comments reformatted, use the `//` style comments or select a region of code then **Source** → **Toggle Comment**. Try this feature out.

Fix any syntax errors before you go on. Look for those little red signs!

## 7 Other Helpful Editing Shortcuts

This section describes some other useful editing techniques that will save you a lot of time.

Create a new `BankAccount` class. For your `BankAccount` class, insert three *fields* (instance variables) for the name of the individual who holds the account, an integer account number and a double for the current balance.

```
private String name;  
private int  accountNum;  
private double balance;
```

### 7.1 Insert Getters and Setters in a Class

After you have inserted your instance variables, Eclipse can automatically insert the getter and setter methods based on your three fields (instance variables). Click on the line where you want the new methods inserted, e.g., choose the blank line above the comment of the `main()`, then select **Source** → **Generate Getters and Setters...**

Select all three fields, click on the box to “Generate method comments” and then press OK.

For the new getter and setter methods, fill in the Javadoc comments with the appropriate information. Remember to select the Javadoc tab near the bottom to see how the Javadoc comment will be formatted.

### 7.2 Insert a Constructor Based on the Fields

Click on the line where you want the new constructor inserted then select **Source** → **Generate Constructor using Fields...**

Select all three fields wanted in the three-parameter constructor then select “Generate constructor comments” and “Omit call to default constructor `super()`”. Then click OK.

Remember to fill in the Javadoc comments with the appropriate information.

### 7.3 Adding Javadoc Comments to a Method

Eclipse has a handy facility for adding Javadoc comments to a method you have already written. For example, enter the following method in your `BankAccount` class.

```
public void printXY(int x, double y) {
    System.out.println("x = " + x + " y = " + y);
}
```

Place the cursor anywhere within the method and select **Source** → **Generate Element Comment**.

```
/**
 * @param x
 * @param y
 */
public void printXY(int x, double y) {
    System.out.println("x = " + x + " y = " + y);
}
```

Note that it has inserted the names of the parameters. All you need to do is add descriptions. Change the comments so that they are meaningful. Here is a suggestion for `main()`.

```
/**
 * Begins program execution.
 *
 * @param args
 *         contains command line arguments
 */
```

### 7.4 Method Name Completion

Suppose you are typing the name of a method in Java's API but you can't remember how to spell it. Eclipse will help you with this. Add a new line to `main()` as follows

```
System.out.prin
```

and stop typing after the "n". Select **Edit** → **Content Assist** → **Default**. Eclipse will offer a list of suggestions. Use the arrow keys to move through the list and select

```
println(String s) void
```

and press Enter. Eclipse has highlighted the `s`, so whatever you type will replace it. Type any string you would like to print.

### 7.5 System.out.println Shortcut

Using `System.out.println` is something that happens so frequently, that Eclipse has a nice shortcut for it. For example, suppose you wanted to print "Hello World!". In `main()` type the following string on a line by itself *with at least one space before*.

```
"Hello World!"
```

Now highlight this string with your mouse and select content assist (**Edit** → **Content Assist** → **Default**). Use the arrow keys to select “sysout — print to standard out” and press Enter. (You can also double click with the mouse.) You will find content assist so useful that you will begin to invoke it with its keyboard shortcut (Control-space).

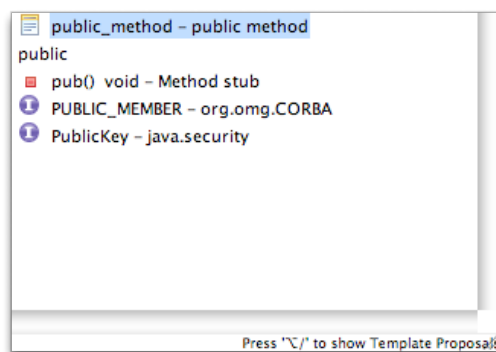
## 7.6 Creating a New Method

Suppose you would like to create the following method.

```
public boolean isZero(int count) {
    return count == 0;
}
```

Complete the following steps.

1. Type just the letters `publ` and invoke content assist.
2. The first choice in the list of options is to create a public method. Double click on it (or press Enter).



3. Eclipse has highlighted the return type. Type the word `boolean` to replace what is there.
4. Press tab and Eclipse will highlight the method name. Type `isZero`.
5. Press tab and Eclipse will move the cursor between the parentheses so you can type the parameter to the method. Enter `int count`.
6. Press tab again and Eclipse will position the cursor so that you are ready to enter the method body. Complete the method and save.

## 7.7 Importing a Java File Into Eclipse

Just copying a Java file into a directory that Eclipse uses is not enough. The file must be imported. To import a Java file into Eclipse, you should create a new Java Project. Then open the white arrow to the left of the project name. Select the “src” line under the project then select **File** → **Import**. Select **General** → **File System** then “Browse”. Find the directory (folder) with the Java file to import. Click **OK**. Select the Java file by clicking its box. Press **Finish** button. The Java file should be imported and can be found in **src** → (**default package**). Double click the Java file’s name and it should appear in Eclipse’s editor window.

## 8 Additional Information on Eclipse

We have explored only a small number of the features of Eclipse. For additional information about Eclipse, look in the **Help** menu or visit the Eclipse web page <http://www.eclipse.org/>.

## 9 Exercise to Hand In

Use the features of Eclipse to write a class `Purse` that represents a coin purse. It will have three integer instance variables named `numNickels`, `numDimes`, and `numQuarters` that contain the number of nickels, dimes and quarters in the purse. The constructor for the class should initialize these to zero.

Create a method called `addNickels()` that has an integer parameter. It will add that many nickels to the purse. The method will have no return value. Create similar methods called `addDimes()` and `addQuarters()`. Finally, create a method called `getTotal()` that will return the value of the coins in the purse. It will have no parameters and it returns a `double`. Document each of your methods using Javadoc comments.

Add the following `main()` method to your `Purse` class.

```
public static void main(String[] args) {
    Purse myPurse = new Purse();
    myPurse.addNickels(3);
    myPurse.addDimes(1);
    myPurse.addQuarters(2);
    double totalValue = myPurse.getTotal();
    System.out.print("The total is ");
    System.out.println(totalValue);
}
```

Run the program and make sure it is working.

## 10 What To Hand In

Run your program using the Run button in the toolbar or use the **Run** menu. Open a text editor, copy and paste the output and save into a file called `output.txt`. You can use `emacs` or the text editor found in **Red Hat** → **Accessories** → **Text Editor**.

Hand in a copy of `Purse.java` and `output.txt`.