CSC 108 Course Information
Welcome to CSC 108. The website is a great source of information and it is required reading, so you should read it regularly. It will contain important announcements about assignments, labs and tests. Lecture notes and saved sessions from DrJava will also be posted on the website.

Who should take this course?
This course is intended for students who have little or no programming experience. It is also suitable for students who have some programming experience, but who do not have a strong object-oriented programming background. (If you do not know what object-oriented programming is, then don’t worry, because you will learn about it in this course.)

Instructors

<table>
<thead>
<tr>
<th>Section</th>
<th>Instructor</th>
<th>Office</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0101</td>
<td>Jennifer Campbell</td>
<td>BA 4260</td>
<td><a href="mailto:campbell@cs.toronto.edu">campbell@cs.toronto.edu</a></td>
<td>416-978-6320</td>
</tr>
<tr>
<td>L5101</td>
<td>Daniel Wigdor</td>
<td>BA 3234</td>
<td><a href="mailto:dwigdor@cdf.toronto.edu">dwigdor@cdf.toronto.edu</a></td>
<td>N/A</td>
</tr>
</tbody>
</table>

Marking Scheme

<table>
<thead>
<tr>
<th>Work</th>
<th>Weight</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labs (10)</td>
<td>20%</td>
<td>Each lab is worth 2%. See below for pre-lab info.</td>
</tr>
<tr>
<td>Assignments (3)</td>
<td>30%</td>
<td>Each assignment is worth 10%. See below for late penalties.</td>
</tr>
<tr>
<td>Midterm</td>
<td>10%</td>
<td>1-hour test during week 6 Wednesday lecture.</td>
</tr>
<tr>
<td>Final exam</td>
<td>40%</td>
<td>You must get 40% or above on the exam to pass the course.</td>
</tr>
</tbody>
</table>

Term Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Dates</th>
<th>Asn Due Dates &amp; Tests</th>
<th>Reminders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5-9 Jan</td>
<td></td>
<td>5 Jan: First day of classes No lab this week</td>
</tr>
<tr>
<td>2</td>
<td>12-16 Jan</td>
<td></td>
<td>Labs start</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>18 Jan: Last day to add courses</td>
</tr>
<tr>
<td>3</td>
<td>19-23 Jan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>26–30 Jan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2-6 Feb</td>
<td>5 Feb: A1 program due</td>
<td>No lab this week</td>
</tr>
<tr>
<td>6</td>
<td>9-13 Feb</td>
<td>11 Feb: Midterm (in lecture)</td>
<td>No lab this week</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>13 Feb: Exam timetable posted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16-20 Feb – Reading Week</td>
</tr>
<tr>
<td>7</td>
<td>23-27 Feb</td>
<td></td>
<td>7 Mar: last day to drop courses</td>
</tr>
<tr>
<td>8</td>
<td>1-5 Mar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8-12 Mar</td>
<td>11 Mar: A2 program due</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>15-19 Mar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>22-26 Mar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>29 Mar–2 Apr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>5-8 Apr</td>
<td>8 Apr: A3 program due</td>
<td>No lab this week</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(No lates accepted)</td>
<td>8 Apr: Last day of classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9 Apr: Good Friday (University Closed)</td>
</tr>
</tbody>
</table>

Late policy for Assignments
There is a penalty for submitting an assignment after the due time (in the formula below, \( h \) is the number of hours late):

\[
\text{percent penalty} = \frac{[h]^2}{48} + [h]
\]

For example, if your assignment is 24 hours late, then the penalty is \( \frac{24^2}{48} + 24 = 36\% \).

**This late penalty does not apply to Assignment 3. No lates will be accepted for Assignment 3. If you submit it late, then you will receive 0/10.

Labs
Each lab consists of a pre-lab exercise and a 2-hour in-lab exercise. You will work in pairs in the lab, although each of you will be expected to complete the pre-lab separately. If you do not complete the pre-lab by 9am Monday in the week of that
lab you will not earn the 2% for the lab.

Academic Offences
Please read the Rules and Regulations from the U of T Calendar (especially the Code of Behaviour on Academic Matters):

http://www.artsandscience.utoronto.ca/ofr/calendar/rules.htm

Here are a couple of general guidelines to help you avoid plagiarism:

- Never look at another student’s assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions.
- The easiest way to avoid plagiarism is to only discuss the assignment with the course TAs and your instructor.

To find out more about “How to Avoid Plagiarism” please see this website:

http://www.cs.toronto.edu/~fpitt/plagiarism.html

That site explains how we catch cheating and what will happen if you are caught cheating.

Lecture Notes We will post lecture notes to you on the website before each lecture. These notes are meant to summarize what we will be learning in class and they are not intended to replace notes that you may take yourself. In fact, we recommend printing the notes and then adding to them, as you attend lecture. We will also post the code that we write in class to the website.

Getting help in CSC 108H
Besides attending lectures, there are several ways to get help in CSC 108H:

- **Closed Labs:** Labs are a great way for you to get some hands-on practice at programming in Java. Your TA will be a graduate student in computer science and you should feel free to ask him or her questions during the lab. You will not only get practice by doing the labs, but you will also get marked on them.

- **Office Hours:** Each week your instructor will make himself or herself available to you for extra help. Stop by office hours to ask questions or to hear questions asked by other students. This is a great way to learn.

- **Newsgroup:** This is a resource where students can post questions and answers. There are some rules about using the newsgroup, the most important of which are:
  1. sign your full name to all posts
  2. do not give information about your assignment solutions in your postings Students are encourage to both ask and answer questions on the newsgroup.

- **E-mail:** If you are having trouble with the course material or if you need extra help, please do not hesitate to contact your instructor. We will answer as soon as possible. Keep in mind that the closer to an assignment due that that you send an e-mail, the longer your wait for a reply is likely to be, due to the large quantity of messages that we receive. Also, please follow these guidelines for email correspondence:
  1. please read the announcements on the course website and the newsgroup to see if your question has already been answered before sending your instructor email
  2. include the course number and section in the title of the e-mail
  3. sign your full name to the e-mail

---

**What is programming?**
Programs are a set of instructions that tell the computer what to do.

Example tasks that a computer program performs are:

- duplicate a file
- draw a circle on the screen
• arrange files alphabetically

A **programmer** is a person who writes computer programs. Programmers are also called **developers**.

**Introduction to DrJava**
In this course, we will be using **DrJava** as our programming environment. This is where we will **program** (write our Java code).

DrJava is available on the CDF-PC environment. A CDF-PC account should have been automatically created for you, when you signed up for this course. You can also install DrJava on your home computer and instructions on how to do this are on the course website.

DrJava is compose of three main panes:

• the upper-left pane contains a list of the pieces of code (called classes) that you have written

• the upper-right pane contains a piece of code (a class) that you have written

• the bottom panel is the interactions pane, where you can try out individual statements without writing an entire program. DrJava is an **interactive interpreter**.

![DrJava Interface](image)

**Java Comments**
// This is a Java comment.
Comments are used by programmers to describe code. The “//” tells Java to ignore that line.

**Java Expressions**
A Java **expression** is a construct that produces values.
For example, Java knows about math.

**Integer expressions**
Type these at the **command prompt** (>) in the interactions pane:
1 + 2
4 - 3
1 + 3 * 2
1 * 3 + 2

Operators have **precedence** and expressions are evaluated as from left to right with higher precedence operators evaluated first.

Consider this expression:
5 / 4

Let’s examine that last answer. 4/5 is 1 with a remainder of 1. So where is the remainder?

Expressions involving integer numbers give integer values.

Try a few more expressions:
4 / 5
8 / 3

The **mod** operator (%) gives the remainder:
5 % 4
4 % 5
8 % 3

**Real number expressions**
Let’s try a few real number expressions:
100.0 - 44.0 3.3 + 2.2 3.3 + 3.9

Note that when the expression 3.3 + 3.9 is evaluated, the value is 7.199999999999999. We would expect the value to be 7.2.

Real numbers in Java are:
- Inexact! These numbers are not real numbers, but they are an approximation.
- These approximations to real numbers are called **floating-point**.

In Java, float-point numbers are called **doubles** (double floating-point precision).

**Mixing int and double**
Try these expressions:
4 / 5.0
4.0 + 3
6 * 1.2

When an expression involves both int and double values, the value generated by the expression is a double.
4.0 / 5
Java tries to keep int values as int values for as long as possible.
Consider the value of:
(12 / 7) + 2.5

**Boolean Expressions**
Java also has a type called **boolean**.

Try these expressions in the interactions pane:
true
calse
true && false // “true and false”
ture || false // “true or false”
true // “not true”
Characters
There is a type in Java called char, which represents characters.

Once again, type these expressions in the interactions pane to see their values:
a would give an error
'a' gives a char must have single quotes around it

Variables
We have been testing out a number of different expressions in Java, using some of Java’s primitive types (int, double, boolean, char).
A variable is a name with an associated type and value.

Variable Declaration
To create a variable we declare it.

A variable declaration looks like:
<type> <variable>;
For example, to declare a variable named x of type double:
double myNumber;

A declaration is a statement: A computer instruction. Expressions are not statements! All simple statements end in a semicolon. Later you will be introduced to more complex statements that do not end in a semicolon.

Variable Initialization
After a variable has been declared, then you can start to use it. What is a variable’s value when it is declared?

Try finding out the value of myNumber:
myNumber

After a variable has been declared, you should initialize it, by assigning it an initial value.

Variable initialization is of the form:
<variable> = <expression>;

For example:
myNumber = 2.7;

Now check to see what the value of x is:
myNumber

You can also declare and initialize a variable in a single statement called an initializing declaration:
int output = 4;

Variable Assignment
It is possible to assign a value to a variable more than once. The previous variable value is overwritten by the new value.

A variable assignment takes the same form as a variable initialization:
<variable> = <expression>;

For example, a variable assignment is:
myNumber = 8.0;
Now check to see what the value of myNumber is:
myNumber

Let’s try using variables:
myNumber + output
output = (1 + 5) * 5;
boolean myBool = true;
myBool
myBool && false
!myBool
myBool = 6; // Why does this result in an error?

**Typecast**
Consider the expression:

```
(12 / 7) + 2.5
```

Earlier we saw that Java evaluated the expression to be a double. What if we want the value of that expression to be an int? We can tell Java to treat an expression as a certain type, by **typecasting** an expression.

For example:

```
(int) ((12 / 7) + 2.5)
```

We can also typecast expressions involving variables, for example:

```
(int) myNumber
(double) output
```

Remember that typecases does not actually changing the value of the variable, but is changes the type of the expression.

**Capitalization**
Type these expressions in the interactions pane:

```
int hello = 8;
Hello
hello
double Hello = 9.8; Hello
hello
```

Capitalization is very important in Java!

**Classes: Non-primitive types**
So far, we have looking at some of Java’s primitive types, including int, double, boolean and char.

Java also has **non-primitive types**, called **classes**.

Let’s start with the class JFrame. A JFrame is a window on the computer screen.

First we tell Java that we are going to use code in the package java.swing.

```
import javax.swing.*;
```

Now we can create an **object** of type JFrame:

```
JFrame myFrame = new JFrame();
// This is an assignment statement.
// The right-hand side is an expression that says to make a new object of type JFrame.
// The expression is evaluated and the “memory address” of the JFrame is stored in myFrame.
```

This is a **method call**:
```
myFrame.show(); // Display the JFrame.
```

A **method call** is a request for an object to perform a particular action. The “.” is an operator, which connects myFrame to show(). This method tells myFrame to display itself on the screen.

Here are some more method calls:
myFrame.setSize(400, 400); // Set the size of the JFrame.
myFrame.setSize(); // Get the size of the JFrame.
myFrame.setTitle();
myFrame.setTitle(“This is the title.”);

Let’s make another JFrame:
JFrame secondFrame = new JFrame();
secondFrame.show();

**Aliasing**

*Aliasing* occurs when 2 variables reference 1 object. For example:
secondFrame = myFrame;
secondFrame.show();
myFrame.show();
myFrame.setSize(50,50);
secondFrame.show();

**Terminology**

Here are a few programming terms:

- **instruction**: An action to be performed, like “add 2 and 3”.
- **Java statement**: An instruction written using the Java language.
- **Java program**: A collection of Java statements.
- **compile**: Translate statements into a language that computers understand.
- **execute**: follow the instructions.

Java programs must be compiled before computers can execute them.