School of Computer Science

CPS109 Course Notes Set 2

Alexander Ferworn Updated Fall 15

Unrelated Facts Worth Remembering

- Get to know the profs. Visit them. Do what they want you to do. Do not remain anonymous!
- At Ryerson [computer science] getting your degree works the same way as treading water while trying to catch gold bricks. Most people can tread water with a bit of practice--that's getting into Rerson in the first place. The trick comes when these sadistic profs. start tossing gold bricks of knowledge at you and you have to catch them and keep treading water. You start to see the problem. You win when you graduate before you sink![i]

Table of Contents

1 Introduction

The following set of topics is designed to introduce you to some straight-ahead code. There are no real tricks associated with any of this stuff.

1.1 Conventions

Over time I have fallen into the rut of using several pieces of standard notation which I use to explain lots of stuff. The notation only has a few parts but you do need to know what it means. Here it is;

Symbol	Meaning	Example
< >	Needs more expansion, or "put your stuff in here"	<variable>
Anything in bold	Java keyword/operator, etc	=, **static**
[]	Optional	**if(**<condition>**)** <statement>**;** [**else** <statement>**;**]
…	There is more code but I'm not going to show you it	… x = 7; …
Syntax	Tells you about some java syntax	*Syntax:* ❑ <variable> = <expression>**;**
Definition	Defines some concept	*Definition:* ❑ static means …
Rules:	Tells you some java rules	*Rules:* ❑ Never code Java while sitting on train tracks.

1.2 Some words about coding style

Follow the coding guidelines in the course text (Chapter 1). The idea behind acquiring a style is very simple. Most people do not simply write a program and forget about it. In fact, most programs take on a life of many years with many people taking on the responsibility of fixing and renewing the code. The problem is if everyone uses a radically different style it becomes very difficult to see what the program was intended to do. A coding style helps to communicate intent to those that follow…and to yourself if you revisit a piece of code after a long time.

A coding style usually provides you a means of showing how the program hangs together by using tabs or spaces to line up elements that are related (white space) over many lines of code. This will become clearer as you learn more Java.

2 Data: Primitive and Object

There are two basic aspects of programming: data and instructions. To work with data, you need to understand ***variables*** and ***types***; to work with instructions, you need to understand *control structures* and *subroutines/methods/procedures*. You'll spend a large part of the course becoming familiar with these concepts. In this document we will concentrate on data.

There are two kinds of data in Java. One is called primitive data. **Primitive data** is handled by facilities built directly into the language. You are probably most familiar with this type of data. For example, Java has facilities for dealing with whole numbers (integers) as you would expect. All primitive data have specific things you can do with them and places where you can use them as defined by

the Java language itself. For example, I might be able to add one integer to another integer and produce a third integer. This is all handled by Java as in any other high level language.

The second kinds of data are objects. Objects are a little more flexible. **Objects** are things that can also take on certain values and be used in certain places but they also have the added dimension of behavior. In a sense they are data that has associated actions defined on it. For example, all strings (zero or more characters put together) in Java are objects and each string you make can be asked how long it is.

3 Variables

A **variable** is just a memory location (or several locations treated as a unit) that has been given a name so that it can be easily referred to and used in a program. The programmer only has to worry about the name; it is the programming language's responsible to keep track of the memory location. But the programmer does need to remember that the name refers to a kind of "box" in memory that can hold a certain "type" of data. Variables--as the name suggests--can change their value.

You can think of a variable as a kind of post office box with a particular address in memory. The box is only so big and can only fit certain things.

> **Definition**: *A variable is a location in memory that has a name and can store a specific type of data. The value stored in the variable can change but the type must always be the same.*

4 Data Types

In Java--and most other languages--a variable has a **type** that indicates what kind of data it can hold. One type of variable might hold integers -- whole numbers such as 3, -7, and 0 -- while another holds Real (floating point) numbers -- numbers with decimal points such as 3.14, -2.7, or 17.0. (Yes, the computer does make a distinction between the integer 17 and the floating-point number 17.0; they actually look quite different inside the computer.)

There could also be types for individual characters ('A', ';', etc.), strings ("Hello", "A string can include many characters", etc.), and less familiar types such as dates, colors, sounds, video, or any other type of data that a program might need to store. At the end of the day all this data is stored in a computer as digital bits (binary) but we can't manipulate it that easily unless we use a bit of abstraction.

We would quickly run into problems if all our calculations had to be in binary. Everyone would have to agree on how binary would represent other things like characters or floating point numbers. To avoid this problem there has been a certain amount of agreement about how binary numbers will be used to represent these things—called types, so you don't have to worry. The thing to remember is that in the background it is all just bits being twiddled.

There are nine fundamental data types in Java that we are interested in at the moment:

Type	example value	range of values	comments
boolean	true	true or false	1 bit
byte	3, -8, 0	-128 to 127	1 byte
short	-30000, 28	-32768 to 32767	2 bytes
int	89, -945, 37865		4 bytes
long	89L, -945L, 5123567876L	-9223372036854775808 to 9223372036854775807	8 bytes
float	89.5f, -32.5f	-3.4E38 to 3.4E38 with 7 significant digits accuracy	4 bytes
double	89.5, -32.5, 87.6E45	-1.7E308 to 1.7E308 with 15 significant digits accuracy	8 bytes
char	'c', '9', 't'	Unicode Characters	2 byte
String	"This is a string"	Unlimited number of ASCII characters	NB: This is not a fundamental data type in java

Strings are a reference or object type, where all the rest are primitive types. However the Java compiler has special support for strings so this sometimes appears not to be the case.

Notice that Java supports 4 types of integers (byte, short, int and long) and 2 types of floating point numbers (float and double).

The boolean data type has only two valid values: **true** and **false**. A boolean variable is used to indicate if a particular condition is true or not. But, it can also be used to represent any situation that has two states (on or off for example).

Characters in Java are another fundamental data type. A "character set" is a list of characters in a particular order. Each programming language supports a

particular character set that defines the valid values for a character variable. Java uses the **Unicode** standard for character representation.

You may be used to the **ASCII** (American Standard Code for Information Interchange) representation in other languages that can represent a character in 1 byte however Java was intended to represent more than the Latin characters used in America and Europe. Unicode has space to represent other character sets but each character must take up 2 bytes to allow this.

Strings can be thought of as a bunch of characters put together. An example of a string literal is:

"*this is a test*"

Everything between the double quotes is part of the string including blanks. You would think that strings are a data type in Java and you would be wrong. In fact there is a String class in java that has a whole bunch of methods available for dealing with them. This will be discussed further in a later section.

4.1 Strong Typing

Java is a language that is **strongly typed**. This means that each variable (or constant) must be associated with a specific **type** for the duration of its existence, and you cannot assign a value of one type to a variable of an incompatible type. The intent is to try and prevent you from creating inadvertent errors by forcing the computer to make an approximation in its conversion.

Conversions are something you have to learn to live with in Computer Science.

To understand this problem let us look at the value one third. One third can be represented exactly by using the integer 1 divided by the integer 3, or 1/3. We all know that 1/3 + 1/3 + 1/3 is equal to 1. However, if we were to actually do the divisions (as anyone with a calculator knows) 1/3 cannot be represented as an integer and must be converted into a real or floating point number which can be represented approximately as 0.3333333. This is an approximation because the 3 keeps repeating and it never stops. If we add 0.3333333 to itself 3 times we get 0.999999. This is not the same as 1. We have lost what is called **precision** or accuracy. Java will not let you unintentionally lose precision thus making this type of error impossible.

Rules:

- Each value in memory must be associated with a specific data type.
- This data type determines what operations we can perform on the data.

5 Identifiers or names

An **identifier** is the equivalent of a name for something in a program. Names are used to refer to many different things. In order to use those things, a programmer must understand the rules for giving names to them and the rules for using the names to work with the things to which they refer. That is, the programmer must understand the syntax and the semantics of names.

In, Java, a name is a sequence of one or more characters. It must begin with a letter and must consist entirely of letters, digits, the underscore character '_', and the dollar sign character '$'. For example, here are some legal names:

```
N   n   rate   x15   a_long_name   time_is_$   HelloWorld
```

Uppercase and lowercase letters are considered to be different, so that HelloWorld, helloworld, HELLOWORLD, and hElloWorLD are all distinct names.

Certain names are reserved for special uses in Java, and cannot be used by the programmer for other purposes. These reserved words include: *class, public, static, if, else, while*, and several dozen other words.

Java is actually pretty liberal about what counts as a letter or a digit. Java uses the Unicode character set, which includes thousands of characters from many different languages and different alphabets, and many of these characters count as letters or digits. However, I will be sticking to what can be typed on a regular English keyboard.

Finally, I'll note that often things are referred to by "compound names" which consist of several ordinary names separated by periods. You've already seen an example: **System.out.println**. The idea here is that things in Java can contain other things. A compound name is a kind of path to an item through one or more levels of containment. The name System.out.println indicates that something called "System" contains something called "out" which in turn contains something called "println".

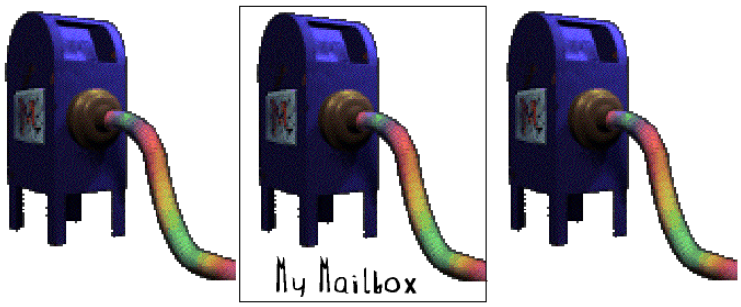
I'll use the term identifier to refer to any name -- single or compound -- that can be used to refer to something in Java.

Rules:

- Must start with the letter A-Z or a-z
- Can use both alphabetic and numeric characters and the underscore character
- There cannot be any spaces in the name
- you cannot use Java keywords as names
- the Java language is case sensitive.

5.1 Declaration

You can't start using a mailbox until it is assigned to you by the post office. The same is true of memory—you must tell Java that you want to use some memory to store something like a variable. You do this when you write the program by declaring space for what you want to store.



Syntax:
The basic syntax for the declaration of a variable in java is

<data type> <variable-name> *[= <expression>] ;*

I my create a variable of type integer by writing the following;

```
int my_mailbox;
```

You can create several variables in the same declaration, if you separate them by commas. For example:

```
double x,y;
char first = 'D', middle = 'J', last = 'E';
int i, j = 17;
```

In the last line, the initialization applies only to j. The value of i is left undefined. (For variables defined outside subroutines, Java will automatically provide i with an initial value of 0 in this case. Variables inside subroutines must be explicitly given values before they are used in an expression.)

Here is an example of a java program, which declares a bunch of different variables and does nothing else.

```
// Example: a bunch of declarations
class BunchDeclares
{
     public static void main (String args[])
     {
          boolean b = true;
```

```
            int low = 1;
            long high = 76L;
            long middle = 74;
            float pi = 3.1415292f;
            double e = 2.71828;
            String s = "Hello World!";
        }
    }
```

5.2 Assignment

Programming languages always have commands for getting data into and out of variables and for performing computations with data. This is accomplished through **assignment**. Assignment happens through the assignment operator or "**=**".

The general form of an assignment is;

<left side> = <right side>;

Where <left side> must be a variable capable of accepting the value produced by <right side>. The value of <right side> will not change.

Consider the following declarations followed by the final statement;

```
float interest;
float principle = 300.0;
…
interest = principal * 0.07;
```

<left side> in this case will take on the value of principal multiplied by 0.07.

syntax

- An assignment statement has the basic syntax of
  <Variable name> **=** <expression>**;**
- On execution, the expression is evaluated and its resulting value is assigned to the variable.
- A variable can only store one value of its declared type, and new values destroy old ones.

5.3 Local variables

Variables defined inside a subroutine (you will have to excuse me, you have probably read that I use subroutine, method, procedure interchangeably…for the

moment they are) are called local variables for that subroutine. You can think of a subroutine as anything contained within curly braces. ({ }).

Declarations exist only inside the subroutine, while it is running, and are completely inaccessible from outside. Variable declarations can occur anywhere inside the subroutine, as long as each variable is declared before it is used in any expression. Some people like to declare all the variables at the beginning of the subroutine. Others like to wait to declare a variable until it is needed and its initial value is known.

Style:

- Declare important variables at the beginning of the subroutine, and use a comment to explain the purpose of each variable.
- Declare "utility variables" which are not important to the overall logic of the subroutine at the point in the subroutine where they are first used.

Here is a simple program using some variables and assignment statements:

```
public class Interest
{
      /*
      This class implements a simple program that
      will compute the amount of interest that is
      earned on $17,000 invested at an interest
      rate of 0.07 for one year.  The interest and
      the value of the investment after one year are
      printed to standard output.*/

      public static void main(String[] args)
      {
            double principal = 17000;
            // the value of the investment
            double rate = 0.07;
            // the annual interest rate
            double interest;
            // interest earned in one year

            interest = principal * rate;
            // compute the interest
            principal = principal + interest;
            // compute value of investment
            // after one year, with interest
            // (Note: The new value replaces
            // the old value of principal.)
```

```
            System.out.print("The interest earned is $");
            System.out.println(interest);
            System.out.print
            ("The value after one year is $");
            System.out.println(principal);
      } // end of main()
} // end of class Interest
```

6 Constants

All the rules that apply to variables apply to constants except that after declaration their value cannot change.

Definition

- constant: a spot in memory which, after initialization, never changes value

Many programs have values that do not change while the program is running, or they might not even change in the real world (pi for example, or the amount of the provincial sales tax) Java provides the facility to declare data items as constants and give them a value.

Rules:

- Constants can only be declared at the top of a class and not as local variables within a method. (This will become clearer when we discuss the anatomy of a Java program)

Syntax:

The basic syntax for the declaration of constants in java is;

final <data type> <constant name> *[***=** *<final value>]***;**

Here is an example program with a constant declared in it.

```
// Example: Declares a bunch of declarations
class MoreDeclares
{
      final String t = "The first declaration";
      public static void main (String args[])
      {
            boolean b = true;
            int low = 1;
            long high = 76L;
            long middle = 74;
            float pi = 3.1415292f;
            double e = 2.71828;
            String s = "Hello World!";
```

```
        }
}
```

7 The Anatomy of a Java Program

Every Java application program must define a subroutine called main, with a definition that takes the form:

```
public static void main(String[] args)
{
      <statements>
}
```

The word public here means that this subroutine can be called from outside the program. This is essential because the Java interpreter actually runs a program by calling its main subroutine. The remainder of the first line is harder to explain; for now, just think of it as part of the required syntax. The definition of the subroutine -- that is, the instructions that say what it does -- consists of a sequence of "statements" enclosed between braces, { and }.

Here, I've used <statements> as a placeholder for the actual statements. I will always use this format: anything that you see in bold text (and in green if your browser supports colored text) is a placeholder that describes what you need to type when you write an actual program.

In Java, a subroutine can't exist by itself. It has to be part of a "class". The main subroutine must be part of a public class that looks like this:

```
public class <program-name>
{
      <optional-variable-declarations-and-methods>
      public static void main(String[] args)
      {
            <statements>
      }
      <optional-variable-declarations-and-methods>
}
```

Note that the name on the first line is the name of the program as well as the name of the class. Thus, the name of the hello java program given previously is Hello. This program should be saved in a file called Hello.java, and when it is compiled, a file named Hello.class will be produced. The resulting class file, Hello.class, contains Java bytecode that can be executed by a Java interpreter.

Also note that a program can contain other methods besides main() as well as things called "variable declarations." You'll learn more about these later.

Style:
We will be dealing a lot with identifiers for variables and constants. I suggest the style that all variables are named with all lowercase characters. Constants should use all UPPERCASE characters (this is not a rule but is good to do to avoid confusion between the two).

8 Strings

Strings are not a native type like int or char in java but are a class…the String class. When you make a string you are really making an object of the string class which is part of the java.lang package.

Unlike other objects, we don't need to use the new constructor (discussed later) with strings, we can simply use a declaration like the following;

```
String name = "Alex"
```

But, since you don't know what a constructor is just remember that you can do the above.

The following code shows two ways of making a String;

```
class stringmaker
{
     public static void main(String[] args)
     {
          String name1 = "Alex";  //way 1
          String name2 = new String("Alex");  //way 2

          System.out.println(name1);
          System.out.println(name2);
     }
}
```

Once a String object contains a value, it cannot be lengthened, shortened, nor can any of its characters change—it is immutable.

There are several methods in the String class that will return new strings that are often the result of modifying the original value.

Here are some useful methods (subroutines) as illustrated in the following program. Don't worry about the term "method" too much. Methods are simply a

functions that become available whenever a value of certain type is created. Let me give you a very harsh introduction. When a factory makes a car and we buy it, in a sense we have bought an object that looks like a car. In fact we know from our experience that each car has a certain number of functions--methods if you like--that can work for us. For example, I can start the car. I can drive the car….I can get the car to tell me how much fuel is left…see if you can relate this to the String you just bought in the example below.

```
class stringmangler
{
     public static void main(String[] args)
     {
          String name = "abcdefghijklxyz";

          System.out.println("The third character is "
                    + name.charAt(3));
          System.out.println("The letter g is at location "
                    + name.indexOf('g'));
          System.out.println("True/False…ends with xyz? "
                    + name.endsWith("xyz"));
          System.out.println("equal to abcdefghijklxyz? "
                    + name.compareTo("abcdefghijklxyz"));
          System.out.println("upper case is: "
                    +  name.toUpperCase());
          System.out.println("Its length is: "
                    + name.length());
     }
}
```

The output of the program is;

```
The third character is d
The letter g is at location 6
True/False...ends with xyz? true
equal to abcdefghijklxyz? 0
upper case is: ABCDEFGHIJKLXYZ
Its length is: 15
```

notes:

- The first line indicates that the third character is d which is true since java begins counting at zero
- The same is true of the second line
- The equal to … line results in zero…this means if you want to check for string equality you must compare the result with zero

- Note how we gain access to the methods within the String class through the use of the "." or dot operator.

9 Operators, Operands and Expressions

definitions

- Operand: something that you can perform an operation on. (eg. 73, var, 9.8)
- Operator: something that performs an operation on an operand (eg. +, -)
- Expression: anything that results in a value usually involving both operators and operands (this is commonly thought of as a calculation (eg. 2+2)

Java follows a well-defined set of rules for governing what operations should be done first (algebraic order of operations or precedence).

You can use round brackets to change the order of operations.

Precedence	Operator	Operation	example
1	+	unary plus	+1
	-	unary minus	-1
2	*	Multiplication	3 *4
	/	Division	3 /4
	%	remainder	3%4
3	+	Addition	2 + 3
	-	Subtraction	3 – 2
	+	String concatenation	"the" + "dog"
4	=	Assignment	Val = 3

Notes:
There is a difference between integer and floating point division. At the end of the following evaluation val will be equal to 0, why?

```
int val;
val = (1/2) + (1/2);
```

[i] Anonymous drunk CS student after a disastrous numerical methods exam