

COMPUTER PROGRAMMING IN JAVA

COLUMBIA UNIVERSITY HIGH SCHOOL SCIENCE HONORS PROGRAM

2007 Oct 12 Sat

1. Review

- Primitive data types
- Operators
- Control flow

Today will be a bunch of mini-tutorials to explore some of the nuances of Java specifically and programming in general.

2. Strings

String manipulation is a very common operation for many applications. Remember that there is also the char datatype, but for our purposes it is much more convenient to use Strings

To concatenate: +

Important note:

Unlike the other primitive types, Strings should NOT be compared using “==”; compare using .equals. Using “==” fails for reasons we'll go into later in the course. The bottom line is:

```
String a = "cat";
String b = "dog";

// Use this!
a.equals(b);

// Don't use this!
boolean b = a == b;
```

There is also .equalsIgnoreCase().

.endsWith(String), .startsWith(String)

charAt(char)

indexOf(String or char), lastIndexOf(String or char)

length()

replace(String old, String new)

substring(int startIndex), substring(int startIndex, int endIndex)

trim()

Note that this returns the changed string, and does not touch the original:

```
String s = " a ";

s.trim();
// Prints _a_ (_ is a space)
System.out.println(s);

s = s.trim();
// Now prints a (with no spaces)
System.out.println(s);
```

toUpperCase(), toLowerCase()

Escape characters:

```
\n    newline
\t    tab
\\    backslash
\"    single double quote
```

3. Methods

In Java, the correct terminology for a function is a method. Methods allow us to organize code in a useful and powerful way.

A method is defined in the following way:

```
public <RETURN_TYPE> <METHOD_NAME>(<ARGS>)
{
    <CODE>
}
```

where:

- <RETURN_TYPE> is one of
 - void
 - a data type (int, boolean, String, etc)
- <ARGS> can be nothing, or a comma-separated list

main is a special method.

4. Code blocks, Semicolons, and Scope

Now that we know about methods, we can talk about code blocks and scope.

A code block is either

- a single statement, terminated with a semicolon, OR
- zero or more statements enclosed with curly braces {}

Note that a statement can be empty:

```
// These are all statements
int a = 1;
System.out.println("Statement.");
;
```

Code blocks are important because they help define scope. Scope can be thought of as where a variable is visible.

- Scopes can enclose other scopes.
- All variables are defined in some scope.
- Each scope can access the variables defined in all of its parent scopes.
- It is an error to define a new variable that already exists in that scope or any parent scope.

Example:

```
// These are all statements
int a = 1;
System.out.println("Statement.");
;
```

- A for loop defines a new scope.
- A method defines a new scope.
- Defining a variable outside of all methods (including main) means it's accessible to all methods in that class.

```
String s = "method1(), method2(), and main method can all access me.";

public void method1()
{
    // No relation to int i in method2!
    int i = 3;
}

public void method2()
{
    // No relation to int i in method1!
    int i = 2;
}

public static void main(String[] args)
```

```
{
    for (int i = 0; i < 10; i ++)
    {
        // this is a new scope, which contains the variable int i
    }

    // i no longer exists here, so it's okay to define a new int i
    int i = 5;
}
```

Going back to control structures, remember that if/else, for, and while all expect code blocks. So that means the following are valid:

```
if (true)
    System.out.println("ok");
else
    System.out.println("also ok");

for (int i = 0; i < 10; i ++)
    System.out.println(i);

while (false)
    System.out.println("This will never execute, but it's valid.");
```

In particular:

```
if (false);
    System.out.println("This always prints");
```

4. Comments & Style

Comments are very important!

- //
- Single line comment
- /* */
- Multiline comment

```
// This is a single line comment. Note that the comment goes ABOVE
// what it's commenting!
// Also note that each line needs to start with two forward slashes.
// This creates a new String called "s" that contains the value
// "I love CS!"
String s = "I love CS!";
```

```
/* Alternatively, you can use a forward slash plus asterisk (no
spaces!) to start a multiline comment and an asterisk plus forward
slash to close it. This is useful for commenting out code you don't
want to execute but don't want to delete. */
/*
int a = 1;
boolean block = false;
String of = "code";
*/
String t = "This is the code I want."
```

Style is also very important!

- Java is a “free-form” language
- Variable name conventions
 - someVariable – correct
 - SomeVariable – incorrect
 - somevariable – incorrect

- `some_variable` – incorrect
- Method name conventions
 - same as for variable names
- Later we'll learn about classes, and their naming conventions
- Tabs
 - braces and blocks

5. Operator promotion rules

- Promotion rules
 - It's okay to make something “bigger”: widening promotions
 - `int -> float`
 - `short -> long`
 - It's bad to make something “smaller”: narrowing promotions
 - `double -> int`
 - `long -> short`
 - String promotions
 - `int, short, long, float, double, char -> String`

Handy operators: `+=`, `-=`

6. Constants

```
final int PI = 3.14;
```

7. Looping over an array

```
int[] array = new int[14];
for (int i = 0; i < array.length; i ++)
{
    ...
}

int i = 0;
while (i < array.length)
{
    ...
    i ++;
}
```

8. Common Compiler Errors

```
A.java:5: cannot find symbol
symbol   : variable a
location: class A
```

- This means you forgot to declare variable “a”, or it might mean that the variable was declared in another scope which is not accessible from the current one.

```
A.java:6: cannot find symbol
symbol   : method printn(int)
```

- This probably means you have a typo in the method name.

9. The Java API

Google and the Java API documentation are your friends. If you don't know or remember how a certain class works, just Google:

```
java 1.5 class <NAME_OF_YOUR_CLASS>
```

Alternatively, the link to the entire documentation is:

<http://java.sun.com/j2se/1.5.0/docs/api/>

Assignments

Easy

Prime number detector

- Description
 - Here is a very simple (and very inefficient) algorithm for determining whether a number n is prime or not:

```
for each number  $i$  between 2 and  $n - 1$ :
    see if the remainder of  $n / i$  is equal to 0
    if it is:
        return not a prime
    else:
        return is a prime
```

- Task
 - Ask the user for a number, and print out whether or not it's a prime.

Medium

Character frequency

- Task
 - Ask the user for a single letter and a sentence, and print out the number of times that letter shows up in the sentence.

Hard

Election simulation

- Description
 - Imagine you are working as a political campaign consultant and you wish to write a computer program to model an election. Your idea is to model the behavior of each individual voter as the product of two random events: (A) They show up to vote, (B) They vote for your candidate. This means that each voter will have two numbers between 0 and 1 associated with them: (A) P -attend = the probability that they show up, (B) P -vote = the probability they vote for your boss.

We can assume that this is a two-candidate race so if P -vote is the probability that a voter will vote for your boss then the probability that they vote for the other guy is $(1 - P\text{-vote})$. Based on values of P -attend and P -vote we may group the voting population into three categories:

- (35% of the voters) Your base (P -attend=0.5, P -vote=0.8)
- (35% of the voters) Your opponent's base (P -attend=0.55, P -vote=0.2)
- (30% of the voters) Swing vote (P -attend=0.4, P -vote=0.5)

There are exactly 5,000 registered voters in your district.

- Task
 - Your job is to write an application that simulates such an election to determine an estimate of the probability that your candidate wins. To do this, have your application simulate each of the 5,000 voters. Count the votes your candidate receives, and print the winner.