CIS 210: Introduction to Computer Science

Instructor: Michal Young
Lab Instructors:
  Megen Brittell, Kals Muthuswamy,
  Arash Shamaei
Fall 2010

Obtaining Course Info

Read the class web page:
  • [http://www.cs.uoregon.edu/classes/10F/cis210](http://www.cs.uoregon.edu/classes/10F/cis210)
  All basic class information is there

Follow the class blog:
  Announcements will appear there first

*Keep current!  It is your responsibility.*
  Suggestion: *Subscribe to email notifications for the blog*

Textbook

*Building Java Programs*
  by Stuart Reges and Marty Stepp

Read assigned chapters *before* lecture
come to class with questions
Experiment!
  try examples from the book, and try variations

Introduction to Computer Science

Programming is an *important part* of computer science

*Important*
  It makes everything else possible.

*But just a part*
  There is much more to computer science.
Q: What is Programming

A: Solving problems
The computer is a tool.
• A carpenter must know how to use a hammer, but knowing how to use a hammer doesn’t make you a carpenter.
A programming language is also a tool.
• You will learn Java. You will also learn to program. Not the same thing!
Programmng is mostly about logical analysis and problem solving

Goals for CIS 210

Learn computer science concepts
Problem solving with computation

General programming skills
• includes designing programs to be understood and modified by humans
• includes testing, debugging

Expressing programs in the Java language
• but the programming concepts apply to other languages

Labs

Lab attendance is mandatory
It counts toward your grade!

Labs cover material not in lecture

It’s your best chance to understand how to solve the homework problems

Getting Help

Labs are excellent opportunities to get help
Instructor and GTFs also hold office hours. We want to see you there!
• But if you skip the lecture, don’t ask me to repeat it in office hours. I won’t do that.
Email is also useful. cis210-help@cs.uoregon.edu
We try to answer quickly, usually within 24 hours.

Don’t wait to the last minute
If the assignment is due in two days, and you are completely lost, I probably can’t help you much.
Collaboration, and its limits

Pair programming is allowed on some assignments
  • Pair programming is done with two people working together at one computer: One driver and one observer. Trade roles often.

Pairs chosen at first lab
  • Let GTF know whom you pair with. Keep a log of meetings.
  • Pair turns in one program.

*Always document contributions of all authors*

Other Collaboration

DO discuss the problems
  Discuss general approaches to solving them. Learn from each other.
  If you rely on ideas from someone else, or somewhere else (e.g., a web site), document it in your solution.

DON’T copy or plagiarize
  Write every line of program code yourself.
  We can tell. We do enforce UO academic honesty policy.

Week 1 project

Goal:
  Learn to edit, compile, and run Java programs
  Learn to use system for turning in work
  Begin learning Java programming language

A “warm-up” project
  They will get harder, and more interesting
  Go to lab to get started

Introduction to Java Programming

A program is a sequence of instructions
The instructions must be written in a “programming language” like Java
  Simpler, stricter, more limited than natural language
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}

Its output:
Hello, world!

This program produces
four lines of output

Compile; Run

source code
Hello.java

compile
byte code
Hello.class

run

table

output

Running a Program

Two steps: Compile, Execute

1. examples > javac Hello.java
2. examples > java Hello
Hello World!

This program produces
four lines of output

examples >

What will really happen

examples > javac Hello.java
Hello.java:6: <identifier> expected
    public static void main( String[] args) {
      ^
Hello.java:6: invalid method declaration; return type required
    public static void main( String[] args) {
      ^
2 errors
Diagnose; repair; repeat

Programming languages have rigid rules
• Expect errors. Lots.
Finally it compiles! Hoorah!
Then it crashes

You will spend a lot of time debugging.
Everybody does.
Learn to do it well. It’s part of your craft.

System.out.println

A statement that prints a line of output
pronounced "print-linn" or “print-line”

Two ways to use System.out.println:
• System.out.println("text");
  Prints the given message as output.

• System.out.println();
  Prints a blank line of output.

Structure of a Java program

public class name {
  public static void main(String[] args) {
    statement;
    statement;
    ...
    statement;
  }
}

method: a named group of statements

statement: a command to be executed

Every executable Java program consists of a class,
that contains a method named main,
• that contains the statements (commands) to be executed.

Names and identifiers

You must give your program a name.

public class HelloWorld {

  Naming convention: capitalize each word (e.g. HelloWorld)
  Your program’s file must match exactly (HelloWorld.java)
  • includes capitalization (Java is "case-sensitive")

identifier: A name given to an item in your program.
  must start with a letter or _ or $
  subsequent characters can be any of those or a number
• legal: _myName TheCure ANSWER_IS_42 $bling$
• illegal: me+u 49ers side-swipe Ph.D’s
## Keywords

**keyword**: An identifier that you cannot use because it already has a reserved meaning in Java.

<table>
<thead>
<tr>
<th>abstract</th>
<th>default</th>
<th>if</th>
<th>private</th>
<th>this</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>do</td>
<td>implements</td>
<td>protected</td>
<td>throw</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>import</td>
<td>return</td>
<td>transient</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>instanceof</td>
<td>short</td>
<td>try</td>
</tr>
<tr>
<td>case</td>
<td>extends</td>
<td>int</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>catch</td>
<td>final</td>
<td>interface</td>
<td>strictfp</td>
<td>volatile</td>
</tr>
<tr>
<td>char</td>
<td>finally</td>
<td>long</td>
<td>super</td>
<td>while</td>
</tr>
<tr>
<td><strong>class</strong></td>
<td>float</td>
<td>native</td>
<td></td>
<td></td>
</tr>
<tr>
<td>const</td>
<td>for</td>
<td>new</td>
<td>switch</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>goto</td>
<td>package</td>
<td>synchronized</td>
<td></td>
</tr>
</tbody>
</table>

## Syntax

**syntax**: The set of legal structures and commands that can be used in a particular language.

Every basic Java statement ends with a semicolon ;
The contents of a class or method occur between { and }

**syntax error (compiler error)**: A problem in the structure of a program that causes the compiler to fail.

- Missing semicolon
- Too many or too few { } braces
- Illegal identifier for class name
- Class and file names do not match

### Syntax error example

```java
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!");
4      }
5  }
```

**Compiler output:**

```
Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
Hello.java:3: ';' expected
```

2 errors

The compiler shows the line number where it found the error.
The error messages can be tough to understand!
Computers used to be bigger and more expensive ... this is part of the Bell Relay Computer

Two big ideas ...

1. Instead of connecting cables, let’s store the program in memory!  
   (stored program machine)

2. Let’s write a program that simulates a more convenient computer!  
   (Universal Turing Machine)
Universal Computing Devices

A computer executes instructions
A program is a collection of instructions

A program can simulate another (more convenient) computer.

Almost any computer can simulate any other (with the right programs).

Virtual (imaginary) Computers

Now it’s simulating a (virtual) Java computer, which executes Java programs.
The computer executes programs expressed in binary machine language

Computer “Science”

Not a conventional science.
Neither is mathematics. That isn’t bad.

In 10 years, all current computers will be obsolete
In 10 years, or 20, or 30, very little of the computer science you learn will be obsolete ... because of universality

Programming and CS

Learning to program is just part of CS
But programmability (universality) is the essence

You must understand programming to understand CS

Java is (just) a reasonable example to start with
Variables and Memory

What’s a variable, really?
What’s a type?

Not like variables in math

In math, \( x = x + 1 \) is just wrong

In Java, \( x = x + 1; \) has a meaning ...

Take the value of \( x \),
add 1 to it,
store the result in \( x \).

What’s a variable?

First try:
A variable is a *name* for a location in memory
(a memory “cell”).

(not quite right ... but it’s a start)

When I write “\( x = 32 \)” I mean “put the value
32 in the memory location named ‘\( x \)’.

What’s memory, really?

Memory cells are one big list,
numbered from zero.
The computer access them by
“address” (number).

Variable ‘\( x \)’ might mean cell #6.
\( x = 33 \) might mean:
put 100001 in cell #6

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01011010110</td>
</tr>
<tr>
<td>1</td>
<td>01011010110</td>
</tr>
<tr>
<td>2</td>
<td>01011010110</td>
</tr>
<tr>
<td>3</td>
<td>01011010110</td>
</tr>
<tr>
<td>4</td>
<td>01011011110</td>
</tr>
<tr>
<td>5</td>
<td>01011011110</td>
</tr>
<tr>
<td>6</td>
<td>01111011110</td>
</tr>
<tr>
<td>7</td>
<td>01011110110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Memory

An address (location) is sent on some of these connectors

Contents go in or out on some of these connectors

Memory Representation

It’s all binary (1’s and 0’s)

What does 01100010₂ mean?

(Trick question ... why?)

01100010₂

Means 62₁₆ (interpreted as an integer)
Or 98₁₀
Or ‘b’ (interpreted as an ASCII character)
Or BOUND (interpreted as an x86 instruction)

Or ... it doesn’t “mean” anything, but we can interpret it several ways, as data or as a program instruction.

What’s a variable (2nd try)

A variable is a named location in memory, with an interpretation for the contents of that memory.

int x;

“x” is the name, “int” is how I interpret the bit patterns. Now I know that 01100010₂ means 98₁₀.
Dynamic types

Some languages (like PHP, Python, Perl, ...) store the type of the variable with the contents, in a “tag”. They don’t require variables to have declared types.

<table>
<thead>
<tr>
<th>Address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>010110101110</td>
</tr>
<tr>
<td>1</td>
<td>01010101110</td>
</tr>
<tr>
<td>2</td>
<td>01011010000</td>
</tr>
<tr>
<td>3</td>
<td>11011010110</td>
</tr>
<tr>
<td>4</td>
<td>01011011110</td>
</tr>
<tr>
<td>5</td>
<td>011010101110</td>
</tr>
<tr>
<td>6</td>
<td>011110101110</td>
</tr>
<tr>
<td>7</td>
<td>010111101110</td>
</tr>
</tbody>
</table>

Tag: it’s an integer
And this is its value

Static types in Java

```java
int x;
```

No need to look at a tag – we will always interpret the value of `x` as an integer, and will never store anything else in the location of `x`.

Java, C, C++, C#, and many other languages do it this way. They have “static” or “compile-time” types.

Some Java types

<table>
<thead>
<tr>
<th>name</th>
<th>meaning</th>
<th>example (literal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integer (represented as 32 bits)</td>
<td>42</td>
</tr>
<tr>
<td>long</td>
<td>integer (represented as 64 bits)</td>
<td>42L</td>
</tr>
<tr>
<td>float</td>
<td>floating point (approximation of real number), 32 bits</td>
<td>42.0f</td>
</tr>
<tr>
<td>double</td>
<td>floating point (approximation of real number), 64 bits</td>
<td>42.0</td>
</tr>
<tr>
<td>String</td>
<td>sequence of characters (text)</td>
<td>“Hello World”</td>
</tr>
<tr>
<td>char</td>
<td>one text character</td>
<td>‘h’</td>
</tr>
</tbody>
</table>

Operations depend on types!

15 + 32 is 47

“15” + “32” is “1532”
What is it?

int x = 15;
int y = 2;
int z;

z = x / y;

What value is in z?

What is it?

double x = 15.0;
double y = 2.0;
double z;

z = x / y;

What value is in z?

What is it?

int x = 15;
int y = 2;
int z;

z = x % y;

What value is in z?

What about these?

int x = 2.0;
double y = 3.53;
float z = 1.83f;
String s = “hello world”;
z = z / y;
y = y % z;
No no no no no nooooooo!

int x = 2.0;    // 2.0 is not an integer
double y = 3.53; // ok
float z = 1.83; // Should be 1.83f
String s = “hello world”;  
z = z / y;    // Warning: precision
s = z;       // Incompatible types

So what’s with this?

int i = 42;     // life, the universe, and everything
float f = 42.0f; // everything else
...
System.out.println(“The answer is ” + i +
    “or, if you prefer, ” + f);

How are we getting away with adding strings, integers, and floating point numbers?

Coercion

“Coercions” are implicit conversions from one type to another

“The answer is ” + 32
32 is converted to the string “32”

OK in printing. Otherwise, usually too confusing.

Casts

Like coercions, but explicit ...
float x = 42.84;
int y;
y = (int) x;      // Convert the value of x to integer
    // and then store in y
System.out.println(y);

What does it print?
“Scope” of a variable

```c
{ ... 
{ ... 
    int x;
    x = 3;
} 
    x = 4; 
}
```

The scope of `x` (up to the end of the current block)

Wrong!

x is out of scope.

Why do variables have limited scope?

Suppose you are designing a new and better programming language ... Java+++ ...

Will variables in your language follow scope rules? Why or why not?

Some reasons for scope

I can (safely) reuse variable names, 
(so my names can be shorter)

I know what I can forget

I can get more error messages from the compiler 
(yes, really that’s a good thing)

Scoping variables

Narrower is better

- Declare variables in as small a scope as practical

Wider scope

```c
static void foo() {
    int x;    
    x = 14;
    y = x / 2;
    ...
}
```

Narrower scope

```c
{ ... 
    int x;    
    x = 14;
    y = x / 2;
    ...
}  
```
Variable names matter

The computer doesn’t care ... but other programmers (and graders!) do

“dollars” and “cents” are better than “x” and “y”

but_this_is_not_really_a_very_good_name
longer ≠ better

Naming conventions in Java

ClassNames use CamelCase
variableNames use camelCase too
but they start with a lower case letter
CONSTANTS_SHOUT
(especially if they have wide scope)

The wider the scope, the more descriptive a variable name needs to be

Summary: What’s a variable?

It names a location in memory
It has a type, which determines how the bits in memory are interpreted, and what operations we can apply to the value
It has a scope (the narrower the better)
Its name should be descriptive (enough) and follow Java naming conventions
Truth and Consequences

The type “boolean”

Logic as Algebra

George Boole, 1815-1864
via Claude Shannon

Basic idea:
Treat “true” and “false” as values
Treat “and”, “or”, “not” as operations
true and x = x
true or x = true

The “boolean” primitive type

Like int, float, etc.
but just two values: true, false

» Note to C and C++ programmers: “true” and “false” are not synonyms for 1 and 0 in Java

boolean b;
b = true;

boolean values

Typically from comparisons:
int i=7; int j = 8;
boolean b = ( i == j ); // false

Combine with “and” (&&), “or” (||), “not” (!)
boolean c = (i < j);
b = b && (i > j) || c;
Algebraic properties of “boolean”

“or” is like addition (commutative, associative)
(x || y || z) == x || (y || z) == x || (z || y)

“and” is like multiplication (commutative, associative)

Multiplication distributes over addition:
(x && (y || z)) == (x && y) || (x && z)

Common Comparisons

== “equal” or same as (the same value)
   (more subtle than it looks: later we will distinguish between “the same thing” and “equal value”)
>
“greater than” (7 > 8, “foo” > “bar”)
< “less than”
>=, <= for ≤, ≥
!= for ≠

Using boolean values

We use boolean (true, false) values to make decisions

if (cost > limit) {
    System.out.println("Sorry, out of money.");
}

Boolean Operators

<table>
<thead>
<tr>
<th>Java</th>
<th>logic, math</th>
<th>electronics</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and, ∧, ·</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>not, ¬</td>
<td>NOT</td>
</tr>
</tbody>
</table>
if ( ... ) { ... } else { ... }

```
int limit = 1000;
int balance = 5000;
int cost = 1001;
if (cost > limit || cost > balance) {
    System.out.println("Sorry, out of money.");
} else {
    balance = balance - cost;
}
```

else if ... (alternative form)

```
if (cost > limit) {
    System.out.println("Limit exceeded");
} else if (cost > balance) {
    System.out.println("Sorry, out of money");
} else {
    balance = balance - cost;
}
```

Nesting

```
if (cost > limit) {
    System.out.println("Charge limit exceeded");
} else {
    if (cost > balance) {
        System.out.println("Sorry, out of money");
    } else {
        balance = balance - cost;
    }
}
```

but watch out for ...

```
if (cost > limit)
    System.out.println("Charge limit exceeded");
else if (cost > balance)
    System.out.println("Limit exceeded");
else
    System.out.println("Ok");
balance = balance - cost;
```

WRONG!
Summary: True or False

Boolean type represents logical values
just two values, true and false
Comparisons create boolean values
i < j, “Ben” < “Jerry”
Create expressions with &&, ||, !
Like arithmetic with numbers (almost)
Use in “if” and “if/else” statements
Control what the program does

Loops, part 1
or, more fun with booleans

‘Round and around and around and around.

— Chubby Checker, 1960, “The Twist”

Example “for” loop

static void drawStars(int nStars) {

    for (int i=1; i <= nStars; ++i) {
        System.out.print('*');
    }
    System.out.println();
}

Pieces of the for loop (1)

for (int i=1; i <= nStars; ++i) {
    do this some number of times
}

Purpose: repeat the part in { ... } some number of times.
    Often a “counting” or “definite” loop.
### Pieces of the for loop (2)

```java
for (int i = 1; 
     i <= nStars; 
     ++i)
{
    do this some number of times
}
```

### More on Loops

If condition never becomes false, the loop will execute "forever"

- An infinite loop will continue until some program fault is reached
- The program may need to be externally terminated
- Make sure when coding a loop that you provide logic to advance the condition toward termination
- Logic may be decrement, dividing by two, multiplying by 3, ...
  - Logic must guarantee that condition is eventually false

**Loops are a type of statement**

- A loop can be selected by an if or if-else
- Loop bodies may contain if, if-else, or other loop statements
- So loops may be nested

### Prospective: Control Flow

**Power tools: We’re moving into algorithm design**

- We can make beautiful furniture, or cut off our hands

We’ll spend a little effort on syntax, and a lot of effort on how to **design** and **reason about** loops and conditions