Intro to Computer Science & Programming

A Cranes Club Initiative
Cranes Club

To create opportunities for professionals of Unificationist background to network and apply their expertise to better serve their communities and the greater society.
Overview

First Half: Fundamentals of CS using Java
- What is computer science?
- Object-oriented programming
  - input/output
- Data structures and algorithms
  - multithreading and GUI
  - network programming
  - Android app
Computer Science

The study of using computers to solve problems. How to solve problems better—more and faster?
Computer

A machine that can perform calculations and operations to accomplish various tasks. Fetch, decode, execute.

Program

A set of instructions that tell the computer what to do.
Programming Language

A specific way to write instructions for the computer, with its own syntax and grammar.

- Compiled
- Interpreted
- Imperative
- Declarative
Compiled vs Interpreted

Compiled: broken down into machine language before execution.

Interpreted: executed by an interpreter, which itself is a running program.
● Pros: performs fast.
● Cons: hardware-specific, compile time.
Pros: dynamic, no compile time.
Cons: lots of overhead, slow performance.
Pros: optimizations can be done on the fly.
Cons: overall still slower than compiled.
Bytecode

Source Program

Compiler

Bytecode

Java Virtual Machine

ARM machine code

Intel machine code

SPARC machine code

AMD machine code
Breaking down a Program

```cpp
int a = 1;
MOV 1, %g2
ST %g2, [%a]

BA 01 02
CF 02 A0
```

C++

SPARC Assembly

machine code hex

machine code binary
Declarative vs Imperative

- **Declarative** describes **what**: statements.
- **Imperative** tells **how**: control flow, state.
- Ex: get the min, max, and average price of a stock over the past week.

```csharp
//Declarative implementation

double[] prices;
prices = (results.Selects(item => item.price).ToArray());
double min = prices.Min();
double max = prices.Max();
double avg = prices.Average();

//Imperative implementation

double min = results[0].price;
double max = results[0].price;
double avg;
double sum = 0;
foreach (TradeRecord record in results)
{
    if (record.price < min) min = record.price;
    if (record.price > max) max = record.price;
    sum += record.price;
}
avg = sum / results.Count;
```
Why Java?

- it is used everywhere—Android, web servers, enterprise systems, desktop applications, etc.
- Most popular language for jobs.
- Will help you learn other languages more easily (C#, Python, etc.)
- But, it’s quite verbose.
Why Java?

3 Billion Devices Run Java

Computers, Printers, Routers, Cell Phones, BlackBerry, Kindle, Parking Meters, Public Transportation Passes, ATMs, Credit Cards, Home Security Systems, Cable Boxes, TVs...
### Developer Jobs per Language in NY

<table>
<thead>
<tr>
<th>Job market by Language</th>
<th>Java</th>
<th>C#</th>
<th>C++</th>
<th>JavaScript</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indeed.com</td>
<td>7964</td>
<td>3197</td>
<td>3573</td>
<td>6126</td>
<td>4711</td>
</tr>
<tr>
<td>Dice.com</td>
<td>1876</td>
<td>763</td>
<td>726</td>
<td>1245</td>
<td>921</td>
</tr>
<tr>
<td>CareerBuilder.com</td>
<td>273</td>
<td>136</td>
<td>84</td>
<td>199</td>
<td>138</td>
</tr>
</tbody>
</table>
Average Salary (source: Glassdoor)

<table>
<thead>
<tr>
<th>Position</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associate Software Engineer</td>
<td>$69,305</td>
</tr>
<tr>
<td>Software Engineer</td>
<td>$90,374</td>
</tr>
<tr>
<td>Senior Software Engineer</td>
<td>$106,575</td>
</tr>
<tr>
<td>Staff Software Engineer</td>
<td>$124,324</td>
</tr>
<tr>
<td>Software Developer</td>
<td>$86,226</td>
</tr>
<tr>
<td>Senior Software Developer</td>
<td>$122,296</td>
</tr>
<tr>
<td>Embedded Software Engineer</td>
<td>$82,739</td>
</tr>
</tbody>
</table>
Object Oriented Design

- objects
- classes
- behaviors
- interfaces
- inheritance
- encapsulation
- composition
- polymorphism

Diagram:

```
living things
  ↓
animal
  ↓
mammal
  ↓
mammal
  ↓
primates
  ↓
hominidae
  ↓
human beings
```

```
plant

insect

bird

amphibian

reptile
```
Overview of Objects and Classes

- **Class**: blueprint defining a set of behaviors (functions, methods) and states (fields, properties). aka Type.
- **Object**: an instance of a class.
- All Java programs are made up of classes and objects.
- Furthermore, all classes descend from the Object class.
The Object Class

Object behaviors (methods):

- clone()
- equals(Object obj)
- getClass()
- hashCode()
- notify()
- notifyAll()
- toString()
- wait()
The Object Class

- The Progenitor Class (the God class!)
- All classes in Java… past, present, and future… can do anything the Object class can do.
Example: Let’s Create a Dog Class

What are some dog behaviors?
- Bark
- Growl
- Whine
- Eat
- Sleep
- Fetch

What are some dog properties?
- Height
- Weight
- Coat
- Color
- Sex
- Temperament
public class Dog {
    int height;
    int weight;
    String coat;
    String color;
    Boolean sex;
    Enum temperament;

    void bark(){}
    void growl(){}
    void whine(){}
    void eat(String meal){}
    void sleep(int hours){}
    void fetch(String object){}
}
A Dog Class Object

- Recall an object is an **instance** of a class.
- Fido, AirBud, Lassie, RinTinTin are all instances of the Dog class (i.e. they are all **Dog objects**):
- They are also Object Class objects.

```java
Dog Fido = new Dog();
Dog AirBud = new Dog();
Dog Lassie = new Dog();
Dog RinTinTin = new Dog();

Fido.bark();
AirBud.fetch("basketball");
Lassie.eat("dog food");
RinTinTin.color = "black";
Fido.toString();
```
**Classic Hello World Example**

```java
public class Main {

    public static void main(String[] args) {
        // write your code here
        System.out.println("Hello Cheon Il Guk!");
    }
}
```

Hello Cheon Il Guk!

Process finished with exit code 0
Syntax and Semantics

- Syntax: the grammatical rules of a language.
- Semantics: the meanings of a language.
- English syntax error: Bear honey the likes eat to.
- English semantic error: Honey likes to eat the bear.
Java Syntax Overview
- Reference Types vs Primitive Types
- Variables
- Arithmetic Operators
- Console I/O
- Control Statements
- Comments
- Keywords
Reference Type vs Primitive Type

- **Reference Type**: descendant of Object
- **Primitive Type**: Simple numerical types. Not descendant of Object.
Reference Type

■ **Reference**: the object’s name.
■ The `new` keyword is required to actually create the object in memory (in the heap).

```
Dog Fido; //object reference only. No Dog object yet.
Fido.bark(); //this will not work!
Fido = new Dog(); //Dog object is now created.
Fido.bark(); //this will work!
```
Primitive Type

- The **new** keyword is not used.
- These objects go on the **stack**.
- Wrapper classes are Reference Types

```java
int x = 10;
Integer x_wrapper = new Integer(x);
x.toString(); //this won't work!
x_wrapper.toString(); //this will work!
```
# Java’s Primitive Types

<table>
<thead>
<tr>
<th>Primitive type</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>char</td>
<td>16-bit</td>
<td>Unicode 0</td>
<td>Unicode 2(^{16}) - 1</td>
</tr>
<tr>
<td>byte</td>
<td>8-bit</td>
<td>-128</td>
<td>+127</td>
</tr>
<tr>
<td>short</td>
<td>16-bit</td>
<td>-2(^{15})</td>
<td>+2(^{15}) — 1</td>
</tr>
<tr>
<td>int</td>
<td>32-bit</td>
<td>-2(^{31})</td>
<td>+2(^{31}) — 1</td>
</tr>
<tr>
<td>long</td>
<td>64-bit</td>
<td>-2(^{63})</td>
<td>+2(^{63}) — 1</td>
</tr>
<tr>
<td>float</td>
<td>32-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
</tr>
<tr>
<td>double</td>
<td>64-bit</td>
<td>IEEE754</td>
<td>IEEE754</td>
</tr>
<tr>
<td>void</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Variables

- Named value.
- Left side = name (identifier)
- Right side = value (literals, constants, expressions)

```
int h = 13;
int i = h * 2 / 4;
h++;  
String word = "Hello";
Dog Pluto = new Dog();
Integer holla = new Integer(h);
```
## Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[] . ()</td>
<td>array access, method call</td>
<td>Fido.bark(); dalmations[100] = “Pongo”;</td>
<td>Left to Right</td>
</tr>
<tr>
<td>++ -- - !</td>
<td>increment, decrement,</td>
<td>weight++; degrees--; degrees = -40; verdict = !GUILTY;</td>
<td>Left to Right, Right to Left</td>
</tr>
<tr>
<td>/ % *</td>
<td>divide, modulo, multiply</td>
<td>gdppc = gdp / pop; odd = number % 2; e = m * c * c;</td>
<td>Left to Right</td>
</tr>
<tr>
<td>+ -</td>
<td>add, subtract</td>
<td>sum = 1 + 1; diff = sum - 1;</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>LT, LTE, GT, GTE</td>
<td>if (Fido.weight &lt; RinTinTin.weight)</td>
<td>Left to Right</td>
</tr>
<tr>
<td>== !=</td>
<td>equality</td>
<td>if (AirBud.height == Lassie.height)</td>
<td>Left to Right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
<td></td>
<td>conditional AND, conditional OR</td>
</tr>
<tr>
<td>? :</td>
<td>ternary, conditional operator</td>
<td>(a ? b) result = true : result = false;</td>
<td>Right to Left</td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>assignment</td>
<td>a += 4; google_stock -= 34.0;</td>
<td>Right to Left</td>
</tr>
</tbody>
</table>
**Console I/O**

- **Output:** use `System.out` object.
- **Input:** use `Scanner` object and `System.in` object.

```java
System.out.println("Hello Cheon Il Guk!");
final double PI = 22.0 / 7.0;
System.out.println(PI);
System.out.format("%.5f\n", PI);
Scanner scanner = new Scanner(System.in);
String sentence = scanner.nextLine();
System.out.println(sentence);
int number = scanner.nextInt();
System.out.println(number);
```

Hello Cheon Il Guk!
3.142857142857143
3.14286
welcome to Intro to Comp Sci and Programming
welcome to Intro to Comp Sci and Programming
1234
1234

Process finished with exit code 0
Conditional Statements

- if/else conditional logic
- Can be nested
- Curly braces { } are important.

```java
if (5 > 6) {
    System.out.println("5 is greater than 6");
} else {
    System.out.println("6 is greater than 5");
}
```

```java
if (PI > 3) {
    if (PI < 2) {
        System.out.println("PI is greater than 3");
        System.out.println("PI is less than 4");
    }
}
```

```java
if (10 > 3) {
    if (4 > 10) {
        System.out.println("10 is greater than 3");
        System.out.println("4 is greater than 10");
    }
}
```

6 is greater than 5
4 is greater than 10

Process finished with exit code 0
Comments

- for documentation purposes
- // for single line comment
- /* */ for multi-line comment

```plaintext
// this is a single line comment

/*
   this is
   a multi-line
   comment!
*/
```
Java Keywords

- reserved by the Java language
- cannot be used for naming things

```java
Dog short = new Dog(); // can't do that!
```
## Java Keywords

<table>
<thead>
<tr>
<th>abstract</th>
<th>continue</th>
<th>for</th>
<th>new</th>
<th>switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert***</td>
<td>default</td>
<td>goto*</td>
<td>package</td>
<td>synchronized</td>
</tr>
<tr>
<td>boolean</td>
<td>do</td>
<td>if</td>
<td>private</td>
<td>this</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>implements</td>
<td>protected</td>
<td>throw</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>import</td>
<td>public</td>
<td>throws</td>
</tr>
<tr>
<td>case</td>
<td>enum****</td>
<td>instanceof</td>
<td>return</td>
<td>transient</td>
</tr>
<tr>
<td>catch</td>
<td>extends</td>
<td>int</td>
<td>short</td>
<td>try</td>
</tr>
<tr>
<td>char</td>
<td>final</td>
<td>interface</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>class</td>
<td>finally</td>
<td>long</td>
<td>strictfp**</td>
<td>volatile</td>
</tr>
<tr>
<td>const*</td>
<td>float</td>
<td>native</td>
<td>super</td>
<td>while</td>
</tr>
</tbody>
</table>
Homework 1

- Tip Calculator
- Due by next week’s class
- For more Java practice, go to http://codingbat.com/
- Now it’s quiz time!!!!!