Java Review Slides

Test Format:
- One page of coding basics
- Four pages of interpretation
- Two pages of coding segments (Two problems will be pulled off of Hackerrank)
- One math proof and one page on program correctness

Hello World Program

```java
// HelloWorld
// Author: Chris Wilcox
// Date:   1/1/2015
// Class:  CS160
// Email:  wilcox@cs.colostate.edu

import java.lang.*;

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Literals

- **Literals** are values that are directly recognized by Java:
  - numbers
    - 237, 10, 9
    - 1.5, 5.8, 99.999
    - 50000L, 13.4F
  - characters
    - ‘a’, ‘z’, ‘0’, ‘$’
  - Strings
    - “hello”, “there”
  - booleans
    - true, false

Java Identifiers

- An **identifier** is a name, such as the name of a variable.
- Identifiers may contain only
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning
- The first character cannot be a digit.
Java Types

- In Java, there are two kinds of data types:
  - **Primitive data types**
    - Hold a single, indivisible piece of data
    - Pre-defined by the language
    - Examples: int, char, double, boolean
  - **Classes (Reference Types)**
    - Hold complex combinations of data
    - Programs may define new classes
    - Examples: String, System, Fraction, Pseudo

### Primitive Types

- **Integer types**: byte, short, int, and long
  - most common: int
- **Floating-point types**: float and double
  - most common: double
- **Character type**: char
- **Boolean type**: boolean

### Sample Expressions

<table>
<thead>
<tr>
<th>Ordinary Math</th>
<th>Java (Preferred Form)</th>
<th>Java (Parenthesized)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rate^2 + delta</td>
<td>rate * rate + delta</td>
<td>(rate * rate) + delta</td>
</tr>
<tr>
<td>2(salary + bonus)</td>
<td>2 * (salary + bonus)</td>
<td>2 * (salary + bonus)</td>
</tr>
<tr>
<td>1 / (time + 3 * mass)</td>
<td>1 / (time + (3 * mass))</td>
<td></td>
</tr>
<tr>
<td>a - 7 / t + 9 * v</td>
<td>(a - 7) / (t + 9 * v)</td>
<td>(a - 7) / (t + (9 * v))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Kind of Value</th>
<th>Memory Used</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Integer</td>
<td>1 byte</td>
<td>−128 to 127</td>
</tr>
<tr>
<td>short</td>
<td>Integer</td>
<td>2 bytes</td>
<td>−32,768 to 32,767</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
<td>4 bytes</td>
<td>−2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>Integer</td>
<td>8 bytes</td>
<td>−9,223,372,036,854,75,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>Floating-point</td>
<td>4 bytes</td>
<td>±3.40282347 × 10^{−38} to ±1.40239846 × 10^{−45}</td>
</tr>
<tr>
<td>double</td>
<td>Floating-point</td>
<td>8 bytes</td>
<td>±1.79769313486231570 × 10^{−308} to ±4.94065645841246544 × 10^{−324}</td>
</tr>
<tr>
<td>char</td>
<td>Single character (Unicode)</td>
<td>2 bytes</td>
<td>All Unicode values from 0 to 65,535</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>1 bit</td>
<td>True or false</td>
</tr>
</tbody>
</table>
Assignment Compatibilities

- A value of each following type can be assigned to a variable of type to the right:
  - byte → short → int → long → float → double
  - This is called a type promotion
  - Have to type cast to assign variables to the left
- You can also assign a value of type char to a variable of type int.
- But you have to type cast to assign from int to char

if Statement with else

- An if statement may have an optional else clause that will only be executed when the condition is false
- Example:
  ```java
  if (wages <= 57600.0)
      tax = 0.124 * wages;
  else
      tax = 0.124 * 57600.0;
  ```

boolean Data Type

- boolean
- A primitive data type that can be set to:
  - true
  - false
- Example:
  ```java
  boolean correct = true;
  ```

Numeric Relational Operators

<table>
<thead>
<tr>
<th>Math</th>
<th>Java</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>=&gt;</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>!=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
**boolean Operators**

- Logical “and” (conjunction)
  - Java symbol `&&`
  - Math symbol `∧`
  - true only when both expressions are true
  
  \[(\text{MIN\_WAGE} \leq \text{wages}) \land (\text{wages} \leq \text{MAX\_WAGE})\]

- Logical inclusive “or” (disjunction)
  - Java symbol `||`
  - Math symbol `∨`
  - true when either or both expressions are true
  
  \[(\text{wages} < \text{MIN\_WAGE}) \lor (\text{wages} > \text{MAX\_WAGE})\]

**Java Logical and Arithmetic Operator Precedence Rules**

1. `!` - (unary)
2. `*` / `%`
3. `+` `-`
4. `<` / `<=` / `>` / `>=`
5. `==` `!=`
6. `^` / `&` / `|`
7. `&&`
8. `||`

**Program example: find divisors**

```java
public class foo {
    public static void main(String[] args) {
        int number = Integer.parseInt(args[0]);
        int divisor = 2;
        while (divisor < number) {
            if (((number % divisor) == 0) {  // Added missing closing parenthesis
                System.out.print(divisor + " ");
            }
            divisor = divisor + 1;
        }
    }
}
```

**Temperature Conversion Program**

```java
System.out.println("\tDEGREES C\tDEGREES F");

for (int cent = 50; cent <= 100; cent++)
{
    double fahr = (9.0 / 5.0) * cent + 32.0;
    System.out.print("\t" + cent);
    System.out.print("\t" + fahr);
}
```
Method Examples

• public double sin(double angle)
• public double cos(double angle)
• public char charAt(int index)
• public int indexOf(char c)
• public static int minimum(int i, int j)
• public static String toLower(String s)
• public int[] getArray()

public and private

• public
  – Can access the class, method, or data by name outside defining class
• private
  – Can access the class, method, or data by name only inside defining class

• Classes generally specified as public
• Instance variables usually are private
• Methods can be public or private

Caution: Pass by value

• What do you expect this to print?

```java
public class PassByValue {
    public static void main(String[] args) {
        int number = 100;
        increment(number);
        System.out.println("Number: " + number);
    }
    public static void increment(int n) {
        n++;
    }
}
```

• The value of the argument is copied, so no change to number! This is true of ALL Primitive Types.

Caution: Pass by “reference”

• What do you expect this to print?

```java
public class PassMyArray {
    public static void main(String[] args) {
        int [] number = {10, 20, 30, 40, 50};
        changeArray(number);
        System.out.println(Arrays.toString(number));
    }
    public static void changeArray(int n) {
        n[0] = 15;
        n[3] = 1;
    }
}
```

• Reference types like arrays and objects pass in a memory location so [15, 20, 40, 1, 50] is printed (notice the formatting for Arrays.toString!)
Arrays

- An array is a set of variables of the same type accessed by their index

```java
int[] day = new int[4];
```

![Day's temperature array](image)

Creating and Accessing Arrays

- Figure 7.1 A common way to visualize an array

![Multidimensional array representation](image)

Creating and Accessing Arrays

- Multidimensional array represented as several one-dimensional arrays
- Given
  ```java
  int[][] table = new int[10][6];
  ```
  - Array table is actually 1 dimensional of type `int[]`
  - It is an array of arrays
  - Important when sequencing through multidimensional array

Java's Representation of Multidimensional Arrays
### File class in Java

- Programmers refer to input/output as "I/O".
- Input is received from the keyboard, mouse, files. output is sent to the console, monitor, files, ...
- The `File` class represents files as objects, and is defined in the `java.io` package.
- Creating a `File` object allows you to get information about a file (on the hard disk or optical drive).
- Creating a `File` object does NOT create a new file on your disk.

```java
File f = new File("example.txt");
if (f.exists() && f.length() > 1000) {
    f.delete();
}
```

### Handling Exceptions

- When doing file I/O, we use `IOException`.

```java
public static void main(String[] args) {
    Scanner scan = null;
    try {
        Scanner scan = new Scanner(new File("input.txt"));
        String firstLine = scan.nextLine();
        ...
    } catch (IOException e) {
        System.out.println("Unable to open input.txt");
        System.exit(-1);
    }
}
```

### Opening a file for writing

- Same story as reading, we must handle exceptions:

```java
public static void main(String[] args) {
    PrintWriter output = null;
    try {
        PrintWriter output = new PrintWriter(new File("output.txt"));
        output.println("Integer number: " + 987654);
        ...
    } catch (IOException e) {
        System.out.println("Unable to write output.txt");
        System.exit(-1);
    }
}
```

### Java Bitwise Operators

- Java has six bitwise operators:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>Bitwise XOR</td>
</tr>
<tr>
<td>~</td>
<td>Bitwise NOT</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>LEFT SHIFT</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>RIGHT SHIFT</td>
</tr>
</tbody>
</table>
Wrapper Classes

- Character, Integer, Double, Arrays, etc.
  - Character.toUpperCase(char c);
  - Integer.parseInt(String s);
  - Double.parseDouble(String s);
  - Arrays.toString(int array[]);
  - Arrays.sort(int array[]);
  - Date date = new Date();

Date Class

- Methods to manipulate dates and times:
- Example code and output:
  String myFormat = “dd/MM/yyyy (HH:mm:ss)”;
  SimpleDateFormat dateFormat = new SimpleDateFormat(myFormat);
  Date date = new Date();
  System.out.println(dateFormat.format(date));

  19/11/2013 (21:17:27)

Enumerated Types

- General Form:
  public enum DayOfWeek { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY }
- Example Usage:
  DayOfWeek day = THURSDAY;
  switch (day) {
    case MONDAY:
    case TUESDAY:
    ...

Classes

- Classes are Reference Types
- Classes are the basis of object-oriented (OO) programming.
- What can classes be used for? Classes have many different uses, for example:
  - Data Structures
  - Code Libraries
  - Java Programs
  - Complex Objects
Static (versus non-Static): Data

- Local data resides in a method, class and instance data resides in the class.
- Data defined in the class can be of two types:
  - Data may belong to the class (only one copy that has the same value for all objects of that class)
  - Data may belong to the object (separate storage and different values for each instance)
- Class data is identified by `static`
- Instance data is not `static`

Static (versus non-Static): Methods

- All methods can access `static` (class) data, no instantiation is required, e.g. `Math.PI`.
- Should call `static` methods with class name, e.g. `Math.sqrt()`, not required within class.
- Must call `non-static` methods with an instance, e.g. `string.length()`.
- Only non-static (instance) methods can access `non-static` data, hidden `this` parameter.
- Using an instance to access `static` (class) data or methods produces a warning.

Constructors

- Q: How to initialize instance data when creating an object? A: Use a constructor.
- Constructors are used to create objects and send them data.
- Always named after the class, arbitrary parameters, but no return value!
- Example:
  ```java
  Scanner s = new Scanner(System.in);
  ```

Pre and Post Conditions (20)

```java
public static int foo(int x) {
    // Precondition: -4 <= x <= 2
    return (x * x + 2 * x - 5);
    // Postcondition -6 <= return <= 3
}
```

f(-5) = 10, f(-4) = 3, f(-3) = -2, f(-2) = -5, f(-1) = -6, f(0) = -5, f(1) = -2, f(2) = 3, f(3) = 10
Loop Invariants (21)

```c
int x = 1, y = 2, z = -5;
while (x <= 5) {
    z += y;
x++;
}
```

// Loop invariants
y = 2, 1 <= x <= 6, -5 <= z <= 5
y = 2, 1 <= x <= 6, z = -5 + 5y  (stronger alternative)

Contrapositive Proof (22)

Contrapositive proof: Use a contrapositive proof to show that if a and b are integers, and 5ab is even, then a is even or b is even. Note: Use as many steps as necessary.

<table>
<thead>
<tr>
<th>Step</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Even(5ab) → Even(a) V Even(b)</td>
</tr>
<tr>
<td>2.</td>
<td>¬(Even(a) V Even(b)) → ¬Even(5ab)</td>
</tr>
<tr>
<td>3.</td>
<td>¬Even(a) ^ ¬Even(b)) → ¬Even(5ab)</td>
</tr>
<tr>
<td>4.</td>
<td>Odd(a) ^ Odd(b) → Odd(5ab)</td>
</tr>
<tr>
<td>5.</td>
<td>a = 2j + 1, b = 2k + 1</td>
</tr>
<tr>
<td>6.</td>
<td>5ab = 5(2j + 1)(2k + 1)</td>
</tr>
<tr>
<td>7.</td>
<td>= (20jk + 10j + 10k + 5)</td>
</tr>
<tr>
<td>8.</td>
<td>= 2(10jk + 5j + 5k + 2) + 1</td>
</tr>
<tr>
<td>9.</td>
<td>Odd(2(10jk + 5j + 5k + 3 + 1)) = true</td>
</tr>
<tr>
<td>10.</td>
<td>(5ab) is odd</td>
</tr>
</tbody>
</table>