C++ -> Java - A High School Teacher's Perspective

David B. Levine and Alyce Brady
(modified by Mike Scott)

Motivation

Question at a College Board Workshop

“How much (and what) will change when I switch from teaching C++ to teaching Java?”

Today’s Approach

Consider a few representative topics in each language

- Show examples
- Make comparisons

Java Primitive Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Characteristics</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bit signed integer</td>
<td>-128 - 127</td>
</tr>
<tr>
<td>short</td>
<td>16 bit signed integer</td>
<td>-32768 - 32767</td>
</tr>
<tr>
<td>int</td>
<td>32 bit signed integer</td>
<td>-2,147,483,648 - 2,147,483,648</td>
</tr>
<tr>
<td>long</td>
<td>64 bit signed integer</td>
<td>-9,223,720,368,547,758,808 - 9,223,720,368,547,758,808</td>
</tr>
<tr>
<td>float</td>
<td>32 bit floating point number</td>
<td>± 1.4E-45 to ± 3.4028235E+38</td>
</tr>
<tr>
<td>double</td>
<td>64 bit floating point number</td>
<td>± 4.9E-324 to ± 1.7976931348623157E+308</td>
</tr>
<tr>
<td>boolean</td>
<td>true or false</td>
<td>NA, note Java booleans cannot be converted to or from other type</td>
</tr>
<tr>
<td>char</td>
<td>16 bit, Unicode</td>
<td>Unicode character, \u0000 to \uFFFF</td>
</tr>
</tbody>
</table>
Java Operators

- Basic Assignment:  =
- Assignment Operators: +=, -=, *=, /=, %=
- Arithmetic Operators: +, -, *, /,
  % (remainder)
  - integer, floating point, and mixed arithmetic and expressions
- Increment and decrement operators: ++, --
- Relational Operators: >, >=, <, <=, ==, !=
- Logical Operators: &&, ||, !
- Bitwise Operators, Ternary Operator

Control Structures in C++
Counting Odd Elements in a 20-element array

```c++
int oddCount = 0;
for (int j=0; j<20; j++)
{
    if (A[j] % 2 == 1)
    {
        oddCount++;
    }
}
```

Control Structures in Java
Counting Odd Elements in a 20-element array

```java
int oddCount = 0;
for (int j=0; j<20; j++)
{
    if (A[j] % 2 == 1)
    {
        oddCount++;
    }
}
```

Variable declarations in C++

```c++
// Primitives
int x;
double d;
bool done;

// Objects
Coin penny;
Dice cube(6);
```
Variable declarations in Java

// Primitives
int x;
double d;
boolean done;

// Objects
Coin penny = new Coin();
Dice cube = new Dice(6);

Key difference in Java

◆ Object Variables are references (pointers) to objects, not objects
◆ Variable declaration merely creates a place to store a reference, not an object
  Dice cube;
◆ Must construct objects explicitly
  Dice cube = new Dice(6);
◆ Can have multiple references to same object - sharing

Pointers in Java

◆ Pointers exist in Java, but in a very restricted form.
  int* intPtr; // syntax error

◆ The programmer does not choose which variables are pointers in Java.
◆ Pointers to primitive data types cannot be declared.
◆ Variables who data type are non primitive (arrays, Strings, all types of objects) are always pointers
◆ Must dynamically allocate objects using the new operator and call to a constructor
  Rectangle rect1 = new Rectangle();
◆ Dereferencing occurs automatically with dot operator . or array access []
Variable Initialization

Java
```java
void test()
{
    int x;
    int y = x * 2;
    // syntax error
}
public class Test
{
    int x;
    // instance variable
    // x set to 0 when
    // object created
}
```  

C++
```cpp
void test()
{
    int x;
    int y = x * 2;
    // syntax error
}
```  

Member function calls in C++

```cpp
penny.flip();
robot.moveTo(newLoc);
```  

Method invocation in Java

```java
penny.flip();
robot.moveTo(newLoc);
```  

Parameter Passing

- **C++**
  - Value (default)
  - Reference (if specified)
  - const Reference (if specified)

- **Java**
  - Value (for primitives)
  - Value (for references!, so has reference like behavior)
Parameter Passing

Java
All parameters are passed by value. Parameters is always a copy of the argument it is sent.

no modifiers on parameters

void test(int x, int y)

C++
Parameters can be passed by value (a copy) or reference (the actual variable) or constant reference (the actual variable, but it cannot be changed.)

no modifier &
const &

Classes and Objects

C++
- public / private / protected data and methods
- Specify interfaces through .h files
- Topic can be ignored (albeit unadvisedly)

Java
- public / private / protected data and methods
- Specify external view through documentation
- Topic CANNOT be ignored (though some try!)
- best to work with Objects from the start

Key difference in Java

- Everything is tied to a class
- Almost everything is tied to an object
- Some data and methods are tied to classes, not objects
  - public static void main(String[] args)
  - Math.abs(int i)
  - System.out.println(StringExpression)

Hello World in C++

#include <iostream.h>

int main()
{
    cout << “Hello world” << endl;
    return 0;
}
Hello World in Java

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

Strings in C++ (AP C++)

```cpp
apstring s;
apstring s1 = "Hello, ";
apstring s2("World");
apstring s3 = s2;
cout << s1 + s3;
```

Strings in Java

```java
String s;
String s1 = "Hello, ";
String s2 = new String("World");
String s3 = s2;
System.out.println(s1 + s3);
```

Strings are Special in Java

- String constants look like those in C/C++
  ```java
  String s1 = "Hello, ";
  ```
- `+` operator is overloaded for Strings - *String concatenation*
  ```java
  System.out.println(s1 + s3);
  ```
I/O (A Key Difference)

---

**C++**
- *cin/cout, streams*
- text-based (graphics are an add-on)
- may be tested on the exam

**Java**
- `System.out.println(StringExpression);`
- graphical i/o: forms, applets, text fields, etc
- industrial-strength class hierarchy
- USE WHATEVER YOU ARE COMFORTABLE WITH!

---

Standard Output

- Done via a method (function) call
  ```java
  int x = 12;
  System.out.print( "x is " + x );
  // no carriage return
  System.out.println("x is still " + x );
  // println adds carriage return
  ```
- Note the + operator.
- If either operand of + is a String the + performs string concatenation.
  - the other operand is automatically converted to a String

---

Standard Input

```java
import java.io.InputStreamReader;
import java.io.BufferedReader;
public class ReadAndEchoNumber
{
    public static void main(String[] args)
    {
        InputStreamReader iReader = new InputStreamReader(System.in);
        BufferedReader bReader = new BufferedReader(iReader);
        System.out.print("Enter a number and press return: ");
        try
        {   String result = bReader.readLine();
            double num = Double.parseDouble(result);
            System.out.println("You entered: " + num);
        }
        catch(Exception e)
        {   System.out.println("An Exception occurred while trying to" + " read and output your number");
            System.out.println("The Exception is " + e.toString() );
        }
    }
}
```

**GACK!!**

---

Input Using a Window

- `JOptionPane` is a built in class from the Java Standard Library
- **Advantages**
  - a standard class
  - students make a window!
- **Disadvantages**
  - calling static methods
  - must still parse numbers
JOptionPane Example

import javax.swing.*;
public class Inputex
{
    public static void main(String[] args){
        String name = JOptionPane.showInputDialog("Please enter your name:");  
        JOptionPane.showMessageDialog(null, "Hello " + name);  
        String ans = JOptionPane.showInputDialog("How old are you " + name + "?");  
        int age = Integer.parseInt(ans);  
        JOptionPane.showMessageDialog(null, name + " , you will be " + (age + 1) + " your next birthday.");
    }
}

Input using a Third Party Class

- Lots of teachers have put together simple IO classes
- A good one is EasyReader / EasyWriter from Skylight Publishing
  - www.skylit.com/javamethods/appxe.html
- Advantages
  - practice using objects
  - easy to use
  - can also read from files
- Disadvantage
  - not standard

EasyReader Example

public class Inputex2
{
    public static void main(String[] args){
        EasyReader keyboard = new EasyReader();
        System.out.print("Please enter your name:");
        String name = keyboard.readLine();
        System.out.println("Hello " + name);
        System.out.print("How old are you " + name + "?" );
        int age = keyboard.readInt();
        System.out.println(name + " , you will be " + (age + 1) + " your next birthday.");
    }
}

Linear Aggregate Structures

- C++ arrays
  - No bounds checking; can’t resize
- C++ vectors (apvector)
  - Bounds checking; resizeable
- Java arrays
  - Bounds checking; can’t resize
- Java ArrayLists
  - Bounds checking; auto-sized; more methods; heterogeneous; fast access through “get”
Arrays

- Java has a built in array type
- square brackets used to denote array variable

```java
int[] list;
list = new int[10]; // elements init to 0
for(int i = 0; i < list.length; i++)
    list[i] = i * i * i;
int x = 12;
list = new int[x]; // old data lost
int[] list2 = {1, 2, 3, 4, 5};
```

- can declare array of any type

```java
String[] stringList = new String[5];
```

## Pointers Again

```java
String[] stringList = new String[5];
```

stringList is an array of 5 String **object variables**

"Variables who type are non primitive (arrays, Strings, all types of objects) are always pointers."

```java
String[] stringList;
```

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
```

## Array Index Out of Bounds

### Java

```java
int x = 20;
int[] list = new int[10];
list[x] = 20;
```

// runtime error
// Exception

### C++

```cpp
int x = 20;
int list[10];
list[x] = 20;
```

// code compiles and runs
```cpp
int *list2 = new int[20];
list2[40] = 100;
delete[] list2;
```

## Linked Structures

- Java has NO pointers??
  Actually nothing but pointers! But it is not the programmer's choice
- Java does have linked structures, e.g. linear linked lists, binary trees
Searching a linked list in C++

```c++
while (p != NULL)
{
    if (key == p->data)
        return p;
    p = p->next;
}
```

Searching a linked list in Java

```java
while (p != null)
{
    if (key == p.data())
        // or key.equals( p.data() )
        // if key is an object
        return p;
    p = p.next();
}
```

Out with the old
(and how it changes)

**C++**
- `const` – more uses than MacGyver’s knife
- Operator overloading
- `delete` – to clean up memory management
- Templates to achieve genericity

**Java**
- `final` – a Pascal/Ada-like use of `const`
- NO operator overloading
- Automatic garbage collection
- Polymorphism and inheritance to achieve genericity

Garbage Collector Used

**Java**

```java
int[] list = new int[10];
```

```c++
int *list2 = new int[20];
```

Delete not part of language. Garbage collector runs when necessary during program execution. Programmers not responsible for memory management. No need to write destructors.

**C++**

```
delete[] list2;
```

No delete causes memory leak. Programmers spend lots of time dealing with memory management issues.
In with the New 
(and what it used to be)

- **C++**
  - Inheritance – It’s there, but it’s a pain and generally not taught
  - Multiple implementations of Abstract Data Types – through classes only

- **Java**
  - Inheritance (extends) – It’s there and it’s easy
  - Multiple implementations of Abstract Data Types - through interfaces and the implements keyword

### One Pet Peeve: The Dreaded Assignment

**Java**
```java
int x = 210;
int y = 2 * 105;
if( x = y )
    x++;
// x = y causes syntax error
// the one catch
boolean test = false;
if(test = true)
    System.out.print('Y');
```

**C++**
```cpp
int x = 210;
int y = 2 * 104;
cout << x << endl;
if( x = y )
    x++;
cout << endl << x;
produces output of
210
209
Huh???!??!
```

### Overall Changes
(one man’s opinion)

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Structures</td>
<td>100%</td>
</tr>
<tr>
<td>Declarations</td>
<td>50%</td>
</tr>
<tr>
<td>Array Structures</td>
<td>10%</td>
</tr>
<tr>
<td>Parameters</td>
<td>20%</td>
</tr>
<tr>
<td>Classes</td>
<td>0%</td>
</tr>
<tr>
<td>Linked Structures</td>
<td>0%</td>
</tr>
<tr>
<td>Memory Mgt.</td>
<td>0%</td>
</tr>
<tr>
<td>Input</td>
<td>0%</td>
</tr>
<tr>
<td>Inheritance, etc.</td>
<td>0%</td>
</tr>
<tr>
<td>Dev. Environment</td>
<td>0%</td>
</tr>
</tbody>
</table>

0 - 50 - 100