CS 211: More Syntax, Memory, Equality

Chris Kauffman

Week 3-1
Logistics

This Week
- Lab 3 exercises
- Some labs consolidated
- Project 1 Due Wednesday: Questions?

Goals Today
- A few course mechanics
- More java syntax
- Memory diagrams

Reading
- BJP Ch 1-7
- Lab Manual chapters
More Mechanics

- Textbook and Practicelt!
- Honors Section
- Tools and DrJava
- Cheating
Array and Function Practice

Good exercises: functions that manipulate arrays
- BJP3 Self-Check 7.28: arrayMystery5
- BJP3 Exercise 7.6: stdev
- BJP3 Exercise 7.12: priceIsRight
- BJP3 Exercise 7.13: longestSortedSequence
Last Time

- Comments
- Statements/Expressions
- Variable Types
  - little types, what about Big types?
- Assignment
- Basic Output (Input?)
- Conditionals (if-else)
- Iteration (loops)
- Aggregate data (arrays, structs, objects, etc)
- Function Declarations
- Library System
Easy Exam Questions to Write

Convert to for

double tol = 1e-4;
double S = 45.0;
double x = 45.0/2;
double err;

err = (S - x*x)*(S - x*x);
while(err > tol){
    x = (x + S/x) / 2.0;
    err = (S - x*x)*(S - x*x);
}

Answers in code pack

Convert to while

int x = 48;
int f = -1;
boolean found = false;

for(int i=x-1;
    i>1 && !found;
    i--)
{
    if(x % i == 0){
        f = i;
        found = true;
    }
}
break exits a loop

```java
int guess, correct = 22;
while(true){
    guess = input.nextInt();
    if(guess == correct){
        System.out.println("You guessed right");
        break;
    }
    System.out.println("You guessed wrong");
}
System.out.println("Game over");
```

There is also a continue which skips to the next loop iteration which is sometimes useful
Array Goodies

Declare and Initialize

```c
int a[] = {1, 2, 3, 4};
```

Initialize Dynamically

```c
int b[];
...
b = new int[]{7, 6, 5};
myFunc(new int[]{3,1,4,1,5,9});
```
Functions

Are parameterized code

- Referred to as **methods** in java jargon
- Give me some stuff (arguments)
- I’ll give you something back (return value)
- Java: specify types for arguments and return
- User **return** to finish function and give value back
  - Immediately ends function (even inside loop)
  - Useful for project 1
Functions

Live inside classes, see FunctionDemo.java

// Sum up an array
public static int sumIntArray(int a[]){
    int sum = 0;
    for(int i=0; i<a.length; i++){
        sum += a[i];
    }
    return sum;
}

For now, use the magic word static for functions

- Omitting static changes the meaning of functions significantly
- We’ll start doing that soon
Legacy of the void

- Sometimes a method gives nothing as an answer.
- Return type is void
- In void methods, return is optional

```java
public static void downHere()
{
    System.out.println("Calling down here");
    // no return required
}

static int aNumber = 0;

public static void maybeIncrease(int myArg)
{
    if (myArg <= 0)
    {
        return; // return immediately
    }
    aNumber++;
    return; // optional return
}
```
Playing with Functions

It’s easy to play with static functions in DrJava’s interactive loop. Make sure to use ClassName.functionName(param,parm2).

Welcome to DrJava. Working directory is ...  
> CountDescents.countDescents(new int[]{}) 
0  
> CountDescents.countDescents(new int[]{3,2,1,5,4,3,2,1}) 
2  
> CountDescents.countDescents(new int[]{3,2,1,2,3}) 
3  
> int x = CountDescents.countDescents(new int[]{1,2,3,4,5}) 
> x  
5  
> int [] a = {5,4,3,3,3,3,3,2,1,5,4,4,4,4,3,2,2,2,1};  
> CountDescents.countDescents(a)  
2
What’s the difference between #1 and #2?

**Defined**

```java
public static void doubler1(int x){
    x = 2*x;
}

public static void doubler2(int x[]){
    x[0] = 2*x[0];
}
```

**Used**

```java
public static void main(String args[]){
    int r = 10;
    int s[] = {20};
    doubler1(r);
    System.out.println(r);
    doubler2(s);
    System.out.println(s[0]);
}
```

▶ Code is in **Doubler.java**
▶ To understand the difference, we need to draw memory diagrams of the function call stack and heap
## Two Kinds of types: Primitive and References

<table>
<thead>
<tr>
<th>Primitives</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Little types are primitives</td>
<td>▶ Big types including types you’ll create</td>
</tr>
<tr>
<td>▶ int, double, char, boolean, long, short, float...</td>
<td>▶ String, Scanner, File, Sauce, Exception, ... And all arrays</td>
</tr>
<tr>
<td>▶ Live directly inside a memory cell</td>
<td>▶ Contents of memory cell refer to another spot in memory where the thing actually resides</td>
</tr>
<tr>
<td>▶ Each primitive type has its own notion of a zero value: know what they are as all arrays are initialized to these values</td>
<td>▶ Usually refer to a heap location</td>
</tr>
<tr>
<td>▶ Only a small number of primitive types, can’t make new ones</td>
<td>▶ Identical to a pointer but operations are limited</td>
</tr>
<tr>
<td></td>
<td>▶ Have a single zero-value: null which points nowhere</td>
</tr>
</tbody>
</table>
Another Tricky Example

What’s the difference? What gets printed?

Defined

```java
public static boolean intEquals1(int x, int y){
    return x==y;
}
```

```java
public static boolean intEquals2(int x[], int y[]){
    return x==y;
}
```

Used

```java
public static void main(String args[]){
    boolean result;
    int a=1, b=1;
    result = intEquals1(a,b);
    System.out.println(result);

    int aa[]={20}, bb[]={20};
    result = intEquals2(aa,bb);
    System.out.println(result);

    result = aa==bb;
    System.out.println(result);
}
```
Equality

== does shallow comparisons: compare the contents of two memory boxes.

- Many times this is not what is desired
- Instead want a deep comparison which compares multiple parts
- For that will typically have x.equals(y) methods
- Can also write static functions that do similar things

Array Equality

Write a function

```java
public static boolean intArrayEquals(int x[], int y[])
```

which checks whether two integer arrays are deeply equal to one another.

Write a function

```java
public static boolean intArrayIdentical(int x[], int y[])
```

which checks whether two integer arrays are the same array.