CS 211: Methods, Memory, Equality

Chris Kauffman

Week 2-1

So far...

- ☒ Comments
- ☒ Statements/Expressions
- ⊟ Variable Types
  - `little` types, what about `Big` types?
- ☒ Assignment
- ⊟ Basic Output (Input?)
- ☒ Conditionals (if-else)
- ☒ Iteration (loops)
- ⊟ Aggregate data (arrays)
- ⊟ Function Declarations (`main`)
- ☐ Library System

Functions / Methods

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

Method Basics

Live inside classes, see `MethodDemo.java`

```
public class MethodDemo{

  // 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

```
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++;
  System.out.println(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 ...
> HighlyComposite.numDivisors(5)
2
> HighlyComposite.numDivisors(6)
4
> HighlyComposite.numDivisors(8)
4
> HighlyComposite.highlyComposite(6)
true
> HighlyComposite.highlyComposite(8)
false
>
```

Early Exit from Code Blocks

- Based on structure of code, may want to end some execution *early*
- `break;` immediately finishes the loop in which it is placed
- `return;` or `return answer;` immediately finishes the method in which it appears

break Exits a Loop

```
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

return Exits a Method

```
// Locate the index at which the integer
// query appears in the array arr; throw
// an exception if query is not present
public static int locate(int [] arr, int query){
  for(int i=0; i<arr.length; i++){
    if(arr[i] == query){
      return i;
    }
  }
  throw
    new RuntimeException("query "+query+" not in array");
}
```

What's the difference between #1 and #2?

Defined

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

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

Used

```
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

Primitives

- Little types are primitives
- `int`, `double`, `char`, `boolean`, `long`, `short`, `float`...
- Live directly inside a memory cell
- Each primitive type has its own notion of a zero value: know what they are as all arrays are initialized to these values
- Only a small number of primitive types, can't make new ones

References

- Big types including types you'll create
- `String`, `Scanner`, `File`, `Sauce`, `Exception`, ... And all arrays
- Contents of memory cell *refer* to another spot in memory where the thing actually resides
- Usually refer to a heap location
- Identical to a pointer but operations are limited
- Have a single zero-value: `null` which points nowhere

Another Tricky Example

What's the difference? What gets printed?

Defined

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

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

Used

```
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

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

which checks whether two integer arrays are *deeply* equal to one another.
Write a function

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

which checks whether two integer arrays are *the same* array.

Array and Function Practice

Good exercises: functions that manipulate arrays

- BJP4 Self-Check 7.28: arrayMystery5
- BJP4 Exercise 7.6: stdev
- BJP4 Exercise 7.12: priceIsRight
- BJP4 Exercise 7.13: longestSortedSequence