Last day of Java

Types, String[] args, overflow, StringBuilder, mysteries
CSCI 110 Fall 2016

Eight primitive types

boolean, byte, char, short, int, long, float, double

Read more:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Demonstration of binary

Integer types

Type	Size (bits)	Minimum Value	Maximum Value
byte	8	-128	127
char	16	0	2^{16}-1
short	16	-2^{15}	2^{15}-1
int	32	-2^{31}	2^{31}-1
long	64	-2^{63}	2^{63}-1

Floating-point types

Type	Size (bits)	Minimum Value	Maximum Value
float	32	2^{-149}	$(2 * 2^{23}) * 2^{127}$
double	64	2^{-1074}	$(2 * 2^{52}) * 2^{1023}$

What's the point of this?

```java
import java.util.Arrays;
class SomeClass {
    public static void main(String[] args) {
        System.out.println(Arrays.toString(args));
    }
}
```

The args parameter

```
import java.util.Arrays;
class SomeClass {
    public static void main(String[] args) {
        System.out.println(Arrays.toString(args));
    }
}
```

The 'args' parameter in main lets a user pass in arguments/data from the command line. Everything in the array is a String, so integer arguments must be converted to ints via Integer.parseInt(), etc.

```
$ javac SomeClass.java
$ java SomeClass hello 123 World
[hello, 123, World]
$ java SomeClass 5 6 7 8 9 END
[5, 6, 7, 8, 9, END]
```

What does this program print?

```
import java.util.Arrays;
class SomeClass {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 65538; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

What does this program print?

```
import java.util.Arrays;
class SomeClass {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 65538; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

WTF??

```
$ javac SomeClass.java
$ java SomeClass
-2147385343
```

Overflow

```
import java.util.Arrays;
class SomeClass {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 65538; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

This is called overflow, when the value of a variable becomes too big for its type.

```
$ javac SomeClass.java
$ java SomeClass
-2147385343
```

This String reverse code is $O(n^2)$. Why?

```
String reverseString(String s) {
    String reversed = "";
    for (int i = s.length() - 1; i >= 0; i--) {
        reversed += s.substring(i, i+1);
    }
    return reversed;
}
```

This String reverse code is O(n^2). Why?

```
String reverseString(String s) {
    String reversed = "";
    for (int i = s.length() - 1; i >= 0; i--) {
        reversed += s.substring(i, i+1);
    }
    return reversed;
}
```

Strings are immutable. Each time we add to `reversed`, we discard the old value to create a new String. For example, say s == "apple".

For i = 4 we add "e" to `reversed` which is now "e".

For i = 3 we add "l", discarding "e" (the old value of `reversed`) and replacing it with "el".

For i = 2 we add "p" to `reversed`, discarding "el" and replacing it with "elp".

And so on...

Use StringBuilder to make this function O(n)

```
String reverseString(String s) {
    StringBuilder builder = new StringBuilder();
    for (int i = s.length() - 1; i >= 0; i--) {
        builder.append(s.substring(i, i+1));
    }
    return builder.toString();
}
```

What does this program print

```
int a = 1000, b = 1000;
System.out.println(a == b);
Integer c = 1000, d = 1000;
System.out.println(c == d);
Integer e = 100, f = 100;
System.out.println(e == f);
```

wat

```
int a = 1000, b = 1000;
System.out.println(a == b);  // true
Integer c = 1000, d = 1000;
System.out.println(c == d);  // false
Integer e = 100, f = 100;
System.out.println(e == f);  // true
```

Java caches the object versions of small numbers

```
int a = 1000, b = 1000;
System.out.println(a == b);  // true
Integer c = 1000, d = 1000;
System.out.println(c == d);  // false
Integer e = 100, f = 100;
System.out.println(e == f);  // true
```

If the value p being boxed is true, false, a byte, or a char in the range \u0000 to \u007f, or an int or short number between -128 and 127 (inclusive), then let r1 and r2 be the results of any two boxing conversions of p. It is always the case that r1 == r2.

http://docs.oracle.com/javase/specs/jls/se7/html/jls-5.html#jls-5.1.7

What does this program print

```
List<Integer> l1 = new ArrayList<Integer>(Arrays.asList(1,2,3));
int v1 = 1;
l1.remove(v1);
System.out.println(l1);

List<Integer> l2 = new ArrayList<Integer>(Arrays.asList(1,2,3));
Integer v2 = 1;
l2.remove(v2);
System.out.println(l2);
```

wat

```
List<Integer> l1 = new ArrayList<Integer>(Arrays.asList(1,2,3));
int v1 = 1;
l1.remove(v1);
System.out.println(l1);  // prints [1, 3]

List<Integer> l2 = new ArrayList<Integer>(Arrays.asList(1,2,3));
Integer v2 = 1;
l2.remove(v2);
System.out.println(l2);  // prints [2, 3]
```

Method overloads

```
List<Integer> l1 = new ArrayList<Integer>(Arrays.asList(1,2,3));
int v1 = 1;
l1.remove(v1);
System.out.println(l1);  // prints [1, 3]

List<Integer> l2 = new ArrayList<Integer>(Arrays.asList(1,2,3));
Integer v2 = 1;
l2.remove(v2);
System.out.println(l2);  // prints [2, 3]
```

The remove() method is overloaded. One takes an index, one takes an object to remove:

```
E remove(int index)
boolean remove(Object o)
```