

CSSE 220

Types, Loops, Strings, Arrays and ArrayLists

Check out ArrayListPractice from SVN

Quick Review: What are Types?

- All variables in Java have a “type”
- Describes the data that can be stored in a variable
 - String – text only
 - short/int/long – whole numbers only
 - float/double – numbers with decimals
 - boolean – true or false
 - char – a single text character
- Classes – Class names are also types, let you define your own, more complex, types

Strings

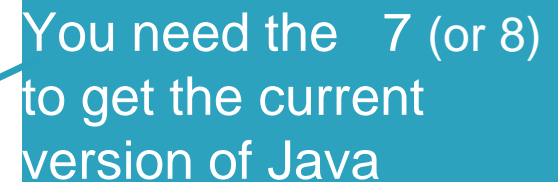
- `String myString = "hello";`
- `String otherString = new String("hello2");`
- Java's way of storing text data
- Has many handy functions like `substring`, `charAt`, etc. that you will slowly learn
- But how do you find out about these cool functions?

Java API Documentation

- What's an API?
 - Application Programming Interface

- The Java API on-line
 - Google for: **java api documentation 7**

You need the 7 (or 8) to get the current version of Java



- Or go to: <http://download.oracle.com/javase/7/docs/api/>

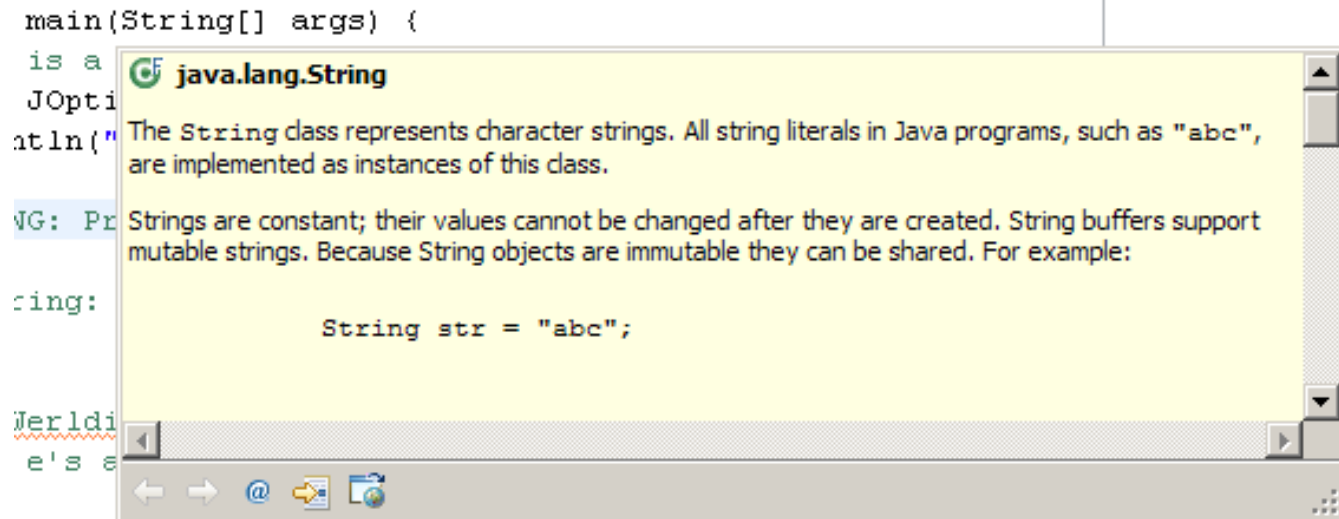
- Also hopefully on your computer at

C:\Program Files\Java\jdk1.7.0_9\docs\api\index.html

Note: Your version may be something other than 7.0_9. We recommend that you bookmark this page in your browser, so you can refer to it quickly, with or without an internet connection.

Java Documentation in Eclipse

- Setting up Java API documentation in Eclipse
 - Should be done already,
- Using the API documentation in Eclipse
 - Hover text
 - Open external documentation (Shift-F2)



Exercise

- If you haven't, finish work on `StringProbs.java`

Review Loops: while & for Loops

- While loop syntax: Similar to Python

```
while (condition) {  
    statements  
}
```

- For loop syntax: Different from Python

```
for (initialization ; condition ; update) {  
    statements  
}
```

In both cases, curly braces optional if only one statement in body; but be careful!

Let's practice some loops

- Go to <http://codingbat.com/java/Warmup-2>
- We'll do countXX together
- Then you do doubleX, stringBits, and (if you have time) stringSplosion

Primitive types

Primitive Type	What It Stores	Range
byte	8-bit integer	-128 to 127
short	16-bit integer	-32,768 to 32,767
int	32-bit integer	-2,147,483,648 to 2,147,483,647
long	64-bit integer	-2^{63} to $2^{63} - 1$
float	32-bit floating-point	6 significant digits (10^{-46} , 10^{38})
double	64-bit floating-point	15 significant digits (10^{-324} , 10^{308})
char	Unicode character	
boolean	Boolean variable	false and true

figure 1.2

The eight primitive types in Java

Most common
number types in Java
code

Gotcha!!!

- int vs. double:
 - `int num1 = 1`
 - `double result = num1 / 2;`
 - //what is result??
- How do we fix this?

Java Loop Examples

- Look at `Investment.java`, `InvestmentTest.java` and `InvestmentRunner.java`
 - Practice using a single `while` loop
 - Study and run the code, then answer quiz questions
- Do the `Rates` exercise in the `Rates.java` file
 - You'll practice using a single `for` loop in that exercise
 - Hint: in `printf`'s format string, use `%%` to display a single `%`

Sentinel Values: A Loop and a Half

- **Sentinel value**—a special input value not part of the data, used to indicate end of data set
 - Enter a quiz score, or Q to quit:
- **A loop and a half**—a loop where the test for termination comes in the middle of the loop
- Examples... (on next slide)

Two Loop-and-a-half Patterns

// Pattern 1

```
boolean done = false;
while (!done) {
    // do some work

    if (condition) {
        done = true;
    } else {
        // do more work
    }
}
```

The variable *done*
here is called a *flag*

// Pattern 2

```
while (true) {
    // do some work

    if (condition) {
        break;
    }

    // do more work
}
```

Arrays- What, When, Why, & How?

- What
 - A special **type** used to hold a set number of items of a specified type
- When
 - Use when you need to store multiple items of the same type
 - Number of items is known and will not change

Arrays- What, When, Why, & How?

- Why
 - Avoids things like int1, int2, int3, int4
 - Avoids repetitive code and frequent updates
- How
 - `Type[] arr = new Type[num];` ← Creates a new array of type Type stored in variable arr
 - An array of 5 Strings (stored in the variable fiveStrings) would look like this:
 - `String[] fiveStrings = new String[5];`

Array Examples Handout

- Form groups of 2
- Look at the Array Examples Handout
- Study how arrays are used and answer the questions in the quiz
 - FIRST PAGE OF QUIZ ONLY

Go to <http://codingbat.com/java/Array-2>

- Work in your groups to solve `fizArray3`, `bigDiff`, `shiftLeft`
- When you finish all 3, call me over to take a look
- If you finish early, try `zeroFront`

Array Types

- ▶ Group a collection of objects under a single name
- ▶ Elements are referred to by their **position**, or *index*, in the collection (0, 1, 2, ...)
- ▶ Syntax for declaring: *ElementType[] name*
- ▶ Declaration examples:
 - A local variable: `double[] averages;`
 - Parameters: `public int max(int[] values) {...}`
 - A field: `private Investment[] mutualFunds;`

Allocating Arrays

- ▶ Syntax for allocating:

`new ElementType[Length]`

- ▶ Creates space to hold values

- ▶ Sets values to defaults

- `0` for number types
- `false` for boolean type
- `null` for object types

- ▶ Examples:

- `double[] polls = new double[50];`
- `int[] elecVotes = new int[50];`
- `Dog[] dogs = new Dog[50];`

Don't forget this step!

This does NOT construct any **Dogs**. It just allocates space for referring to **Dogs** (all the **Dogs** start out as *null*)

Reading and Writing Array Elements

▶ Reading:

- `double exp = polls[42] * elecVotes[42];`

Sets the value in
slot 37.

Reads the element with
index 42.

▶ Writing:

- `elecVotes[37] = 11;`

▶ Index numbers run from 0 to array length – 1

▶ Getting array length: `elecVotes.length`

No parentheses, array length
is (like) a field

Arrays: Comparison Shopping

Arrays...	Java	Python lists
<i>have fixed length</i>	<i>yes</i>	<i>no</i>
<i>are initialized to default values</i>	<i>yes</i>	<i>n/a</i>
<i>track their own length</i>	<i>yes</i>	<i>yes</i>
<i>trying to access "out of bounds" stops program before worse things happen</i>	<i>yes</i>	<i>yes</i>

ArrayList- What, When, Why, & How?

- What
 - A class in a Java library used to hold a collection of items of a specified type
 - Allows variable number of items
 - Fast random access
- When
 - Use when you need to store multiple items of the same type
 - Number of items is not known/will change

ArrayList- What, When, Why, & How?

- Why
 - Fast random access
 - Allows length changes, cannot do this with an array
- How
 - `ArrayList<Type> arl = new ArrayList<Type>();`
 - Creates a new ArrayList of type Type stored in variable arl

ArrayList Examples Handout

- Look at the `ArrayList` section of the examples handout
- Study how `ArrayLists` are used and answer the questions in the quiz
- Then solve the 3 problems in `ArrayListPractice` (you downloaded it from SVN)
- When you finish, call me over to take a look

What if we don't know how many elements there will be?

▶ **ArrayLists** to the rescue

▶ Example:

Element type

```
graph TD; ET[Element type] --> C1[ArrayList<State> states = new ArrayList<State>()]; ET --> C2[states.add(new State("Indiana", 11, .484, .497));]; VT[Variable type] --> C1; AEL[Adds new element to end of list] --> C2; CNEL[Constructs new, empty list] --> C1;
```

```
◦ ArrayList<State> states = new ArrayList<State>();
```

Variable type

Constructs new, empty list

```
◦ states.add(new State("Indiana", 11, .484, .497));
```

Adds new element to end of list

▶ **ArrayList** is a *generic class*

- Type in <brackets> is called a *type parameter*

ArrayList Gotchas

- Type parameter can't be a primitive type
 - Not: `ArrayList<int> runs;`
 - But: `ArrayList<Integer> runs;`
- Use ***get*** method to read elements
 - Not: `runs[12]`
 - But: `runs.get(12)`
- Use **`size()`** not **`length`**
 - Not: `runs.length`
 - But: `runs.size()`

Lots of Ways to Add to List

- ▶ Add to end:
 - `victories.add(new WorldSeries(2011));`
- ▶ Overwrite existing element:
 - `victories.set(0,new WorldSeries(1907));`
- ▶ Insert in the middle:
 - `victories.add(1, new WorldSeries(1908));`
 - Pushes elements at indexes 1 and higher up one
- ▶ Can also remove:
 - `victories.remove(victories.size() - 1)`

So, what's the deal with primitive types?

▶ Problem:

- ArrayList's only hold objects
- Primitive types aren't objects

▶ Solution:

- *Wrapper classes*—instances are used to “turn” primitive types into objects
- Primitive value is stored in a field inside the object

Primitive	Wrapper
<i>byte</i>	<i>Byte</i>
<i>boolean</i>	<i>Boolean</i>
<i>char</i>	<i>Character</i>
<i>double</i>	<i>Double</i>
<i>float</i>	<i>Float</i>
<i>int</i>	<i>Integer</i>
<i>long</i>	<i>Long</i>
<i>short</i>	<i>Short</i>

Auto-boxing Makes Wrappers Easy

- ▶ Auto-boxing: automatically enclosing a primitive type in a wrapper object when needed
- ▶ Example:
 - You write: `Integer m = 6;`
 - Java does: `Integer m = new Integer(6);`
 - You write: `Integer answer = m * 7;`
 - Java does: `int temp = m.intValue() * 7;`
`Integer answer = new Integer(temp);`

Auto-boxing Lets Us Use ArrayLists with Primitive Types

- ▶ Just have to remember to use wrapper class for list element type
- ▶ Example:
 - `ArrayList<Integer> runs =`
 `new ArrayList<Integer>();`
 `runs.add(9); // 9 is auto-boxed`
 - `int r = runs.get(0); // result is unboxed`

Enhanced For Loop and Arrays

▶ Old school

```
double scores[] = ...  
double sum = 0.0;  
for (int i=0; i < scores.length; i++) {  
    sum += scores[i];  
}
```

▶ New, whiz-bang, enhanced for loop

```
double scores[] = ...  
double sum = 0.0;  
for (double score : scores) {  
    sum += score;  
}
```

Say "in"

- No index variable (easy, but limited in 2 respects)
- Gives a name (score here) to each element

Enhanced For and ArrayList's

```
▶ ArrayList<State> states = ...  
int total = 0;  
for (State state : states) {  
    total += state.getElectoralVotes();  
}
```


Work Time

- Finish all the in-class material exercises if you haven't yet
- Work on TwelveProblems