Recap: Objects vs. Primitives

- References: animals = new String();
  - Calls the String constructor.
  - animal is a reference to the String object

- Direct storage: animal = new String("kangaroo");
  - Stores the string directly in animal

Recap: Objects vs. Primitives

- Each Animal is unique, but all are Students
  - You each have names, ID numbers, etc.

Recap: Objects Belong to Classes

- Just as 1, 2, and 3 are all integers, you are all objects of the class UBCStudent!
  - You each have names, ID numbers, etc.
  - Each is unique person, but all are students

Recap: Constructors and Parameters

- Many classes have more than one constructor, taking different parameters
- Use API docs to pick which one to use based on what initial data you have

Recap: Constructors and Parameters

- Class: complex data type
  - includes both data and operations
  - programmers can define new classes
    - many predefined classes in libraries
  - Method: operations defined within class
    - internal details hidden, you only know result
  - Object: instance of class
    - entity you can manipulate in your program

Recap: API Documentation

- Online Java library documentation at http://java.sun.com/javase/6/docs/api/
  - Textbook alone is only part of the story
  - Let's take a look!
  - Everything we need to know: critical details
  - And often many things far beyond current need

Recap: Return Values

- Methods can have return values
- Example: charAt method result
  - Return value, the character 'n', is stored in

Recap: Methods and Parameters

- Methods are how objects are manipulated
  - Pass information to methods with parameters
  - includes inputs to method call
  - Tell charAt method which character in the String object we're interested in

Recap: Classes, Methods, Objects

- Class: complex data type
  - Includes both data and operations
  - Programmers can define new classes
    - Many predefined classes in libraries
  - Method: operations defined within class
    - Internal details hidden, you only know the result
  - Object: instance of class
    - Entity you can manipulate in your program

Recap: Declare vs. Construct Object

- Variable declaration does not create object
  - Creates object reference
  - Constructor and new operator creates object somewhere in memory
    - Constructors can pass initial data to object
  - Assignment binds object reference to created object
    - Assigns address of object location to variable

Recap: Objects vs. Primitives

- References vs. direct storage

Recap: More String Methods

- public String replace(char oldChar, char newChar);
  - Returns a new string object where all instances of oldChar have been changed into newChar.

Recap: Some Available String Methods

- public String toUpperCase();
  - Returns a new String object identical to this object, all the characters converted to upper case.

Recap: Objects vs. Primitives

- References vs. direct storage

Recap: Classes, Methods, Objects

- Primitive Types vs. Classes

<table>
<thead>
<tr>
<th>Primitive Types</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-defined in Java</td>
<td>Written by other programmers or by you</td>
</tr>
<tr>
<td>Simplest things, e.g. int</td>
<td>Can be arbitrarily complex</td>
</tr>
<tr>
<td>Operators: +, *, etc.</td>
<td>Methods</td>
</tr>
<tr>
<td>Values belong to types. E.g., 3 is an int. 3.14159 is a double</td>
<td>Objects belong to classes. E.g., you are a UBC Student</td>
</tr>
</tbody>
</table>

Objects Belong to Classes

- Just as 1, 2, and 3 are all integers, you are all objects of the class UBCStudent!
  - You each have names, ID numbers, etc.
  - Each is unique person, but all are students
  - Social organizations example:
    - Ballroom Dance Club
    - Ski Club
    - CSSS
    - etc.
  - Sometimes called "instances" of a class.
Class Libraries
- Before making new class yourself, check to see if someone else did it already
- Libraries written by other programmers
- Many built into Java
- BigInteger (java.math.BigInteger) lets you compute with arbitrarily big integers.
- Date (java.util.Date) lets you get the current time.
- Calendar (java.util.Calendar) does fancy date computations.

Example: BigInteger
```java
import java.math.BigInteger;
// Tell Java to use standard BigInteger package
public class BigQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("111222333444555666777888999");
        BigQuick stockOptions = new BigQuick("123456789012345678901234567890");
        BigInteger optionCompensation = stockOptions.multiply(salary);  
        BigInteger profitPerShare = new BigInteger("131458");
        BigInteger totalCompensation = salary.add(optionCompensation);
        System.out.println("Total Compensation = $" + totalCompensation.tostring());
    }
}
```

Biginteger Constructors
```java
import java.math.BigInteger;
// Tell Java to use standard BigInteger package
public class BigQuick {
    public static void main(String[] args) {
        BigInteger salary = new BigInteger("111222333444555666777888999");
        BigQuick stockOptions = new BigQuick("123456789012345678901234567890");
        BigInteger optionCompensation = stockOptions.multiply(salary);  
        BigInteger profitPerShare = new BigInteger("131458");
        BigInteger totalCompensation = salary.add(optionCompensation);
        System.out.println("Total Compensation = $" + totalCompensation.tostring());
    }
}
```

Literals
- With the primitive types, how do you create values with that type?
  - E.g., how do we create integer values?
    1. You type some digits, like 3, or 42
    2. You combine integer-valued things with operators that work on integers, e.g.,
      3+42*(a-b)
- A bunch of digits are an integer literal.
- It's the basic way to create an integer value

More Literals
- How about a value of type double?
  1. You type a bunch of digits with a decimal point, and optionally the letter e or E followed by an exponent
  2. You can combine doubles with operators that work on doubles.

Those are literals!

Long Literals
- How about values of type long?
  1. You type a bunch of digits followed by the letter l or L
  2. You combine previously created longs

Literal for Classes?
- Classes are like primitive types, except they can be defined any way you like, and they can be much more complex.
- How to create a value (an object) of a given class?
  1. Invent some way to type a literal???
  2. Operators that create objects of that class (methods).

Using Constructors
- Use a constructor just as you'd use a literal. Example:
  - For the int type:
    int a = 3;
  - For the BigInteger class:
    BigInteger a = new BigInteger("3");

Primitive Types vs. Classes
```text
<table>
<thead>
<tr>
<th>Primitive Types</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-defined in Java</td>
<td>Written by other programmers or by you</td>
</tr>
<tr>
<td>Simplest things, e.g., int</td>
<td>Can be arbitrarily complex</td>
</tr>
<tr>
<td>Operators +, -,...</td>
<td>Methods</td>
</tr>
<tr>
<td>Values belong to types. E.g., 3 is an int, 3.14159 is a double</td>
<td>Objects belong to classes E.g., you are a UBC Student</td>
</tr>
<tr>
<td>Literals</td>
<td>Constructors</td>
</tr>
</tbody>
</table>
```

What about String?
- Is String a primitive type? Is it a class?
- String is a class, but it's a special class!
  - Automatically imported
  - Built-in literals, e.g., "This is a String literal."
  - + operator for concatenation
  - But it also has many other methods, that you can call, just like for any ordinary class...

String Example – Constructor Syntax
```java
public class StringTest {
    public static void main(String[] args) {
        String firstName;
        String lastName = new String("Kermit");
        string.firstName = new String("The Frog");
        System.out.println("I am not "+ firstName + "+ lastName);
    }
}
```
String Example - Literal Syntax

```java
public class StringTest{
    public static void main (String[] args)
    {
        String firstname;
        String lastname;
        firstname = "Kermit";
        lastname = "the Frog";
        System.out.println("I am not " + firstname + " " + lastname);
    }
}
```

String is the only class that supports both literals and constructors!

### Escape Characters
- How can you make a String that has quotes?
  - String foo = "oh so cool";
  - String bar = "oh so \"cool\", more so";
- Escape character: backslash
  - general principle

### Keyboard Input
- Want to type on keyboard and have Java program read in what we type
  - store it in variable to use later
- Want class to do this
  - build our own?
  - find existing standard Java class library?
  - find existing library distributed by somebody else?
- Scanner class does the trick
  - java.util.Scanner
  - nicer than System.in, the analog of System.out

### Scanner Class Example
```java
import java.util.Scanner;
public class Echo{    public static void main (String[] args)    {        String message;  // Declare string variable to store what user types in
        System.out.println ("Enter a line of text: ");
        Scanner scan = new Scanner (System.in);
        message = scan.nextLine();
        System.out.println ("You entered: " + message);    }}
Scanner Class Example
```

- Import Scanner from java.util package
- Use Scanner constructor method to create new Scanner object named scan
  - could be named anything, like keyboardStuff or foo
- Scanner class example
  - Collections of related classes grouped into packages
  - tell Java which packages to keep track of with import statement
  - again, check API to find which package contains desired class
  - No need to import String, System.out because core java.lang packages automatically imported

### Importing Packages
```java
import java.util.Scanner;
public class Echo{    public static void main (String[] args)    {        String message;  // Declare string variable to store what user types in
        System.out.println ("Enter a line of text: ");
        Scanner scan = new Scanner (System.in);
        message = scan.nextLine();
        System.out.println ("You entered: " + message);    }}
Scanner Class Example
```

- Prompt user for input
- nextLine method reads all input until end of line
- returns it as one long string of characters

### Scanner Class Example
```java
import java.util.Scanner;
public class Echo{    public static void main (String[] args)    {        String message;  // Declare string variable to store what user types in
        System.out.println ("Enter a line of text: ");
        Scanner scan = new Scanner (System.in);
        message = scan.nextLine();
        System.out.println ("You entered: " + message);    }}
Scanner Class Example
```

- Let's try running it
- Print out the message on the display