Chapter 3: Using Classes and Objects

CS 121

Department of Computer Science
College of Engineering
Boise State University

September 5, 2016

Chapter 3 Topics

- Part 0: Intro to Object-Oriented Programming

  Go to part 0
- Part 1: Creating Java objects

  Go to part 1
- Part 2: The `Graphics` class

  Go to part 2
- Part 3: The `Random` class

  Go to part 3
- Part 4: Formatting output

  Go to part 4
- Part 5: The `Math` class

  Go to part 5
- Part 6: The `String` class

  Go to part 6
- Part 7: Wrapper classes and autoboxing

  Go to part 7

Brief Intro to Object-Oriented Programming

- Java is an object-oriented programming language.
- Objects can be used to represent real-world things.
- Objects have state and behaviors.

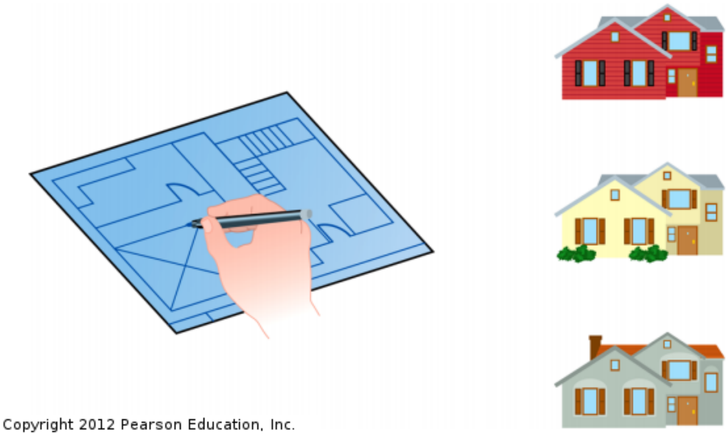
  - Dog
    - state: name, breed, color, age, hungry, etc.
    - behavior: walk, run, bark, lick, fetch
  - String `"Hello World!"`
    - state: length, characters
    - behavior: get length, equals, sub-string, compare to, to upper case, etc.

Classes

- Objects are defined by classes.
- Multiple objects can be created from the same class.
- Variables represent the object's state (attributes, properties).
- Methods define behaviors (functions, actions).

Classes and Objects

- We can think of a class as the blueprint of an object.
- One blueprint to create several similar, but different, houses.



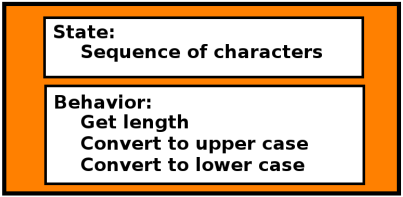
Copyright 2012 Pearson Education, Inc.

Classes and Objects

- An object is an instance of a class.
- Objects are encapsulated, meaning the state and behaviours are wrapped together as a single unit.
- Encapsulation is one of the four fundamental Object-Oriented Programming (OOP) concepts. The other three are inheritance, polymorphism, and abstraction. (You will see these in CS 221).

Encapsulation

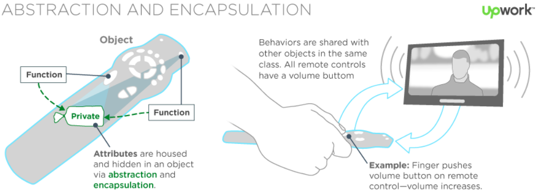
- A visual example of encapsulation of the `String` object.



State:
Sequence of characters

Behavior:
Get length
Convert to upper case
Convert to lower case

- An analogy of a "real-world" object that uses encapsulation.


ABSTRACTION AND ENCAPSULATION
upwork
Object
Function
Private
Function
Attributes are housed and hidden in an object via abstraction and encapsulation.
Behaviors are shared with other objects in the same class. All remote controls have a volume buttom
Example: Finger pushes volume button on remote control—volume increases.

Using Classes and Objects

- For now, we will be *using* classes that other people have created.
- After we grasp the basic concepts of how to use them, then we will see how to write our own classes.

The Java API

- The Java API is the standard class library that provides a large collection of pre-built classes that we can use in our programs.
- API = **A**pplication **P**rogramming **I**nterface
- Before writing our own classes, we will practice using several classes that are part of the Java API.
- The classes of the Java API are organized into packages. Java comes with hundreds of packages and tens of thousands more can be obtained from third-party vendors.
- Java API docs: `http://docs.oracle.com/javase/8/docs/api/`

Selected Java Packages

Package	Provides
`java.lang`	Fundamental classes
`java.util`	Various useful utility classes
`java.io`	Classes for variety of input/output functions
`java.awt`	Classes for creating graphical user interfaces and graphics
`java.swing`	Lightweight user interfaces that extend AWT capabilities
`java.net`	Networking operations
`java.security`	Encryption and decryption

Import Declarations

- When you want to use a class from a Java API package, you need to import the package.

```
import java.awt.Graphics;
```

- To import *all* classes in a package, you can use the wild card character (*).

```
import java.awt.*;
```

- All classes in the `java.lang` package are automatically imported into all programs.
    - This includes `String` and `System` (among others)

Declaring and Instantiating Objects

- We must declare and initialize our objects before we can use them.
- This is very similar to what we did with primitive data type variables, except we must also instantiate our objects.

```
int courseNumber = 121;
Scanner kbd = new Scanner(System.in);
```

- For example, we use the `new` operator to instantiate a new `Scanner` object, which is an instance of the `Scanner` class.
- If we don't use the `new` operator, and just declare an object, this does not create an instance of the class.

```
Scanner kbd;    // this variable refers to
                // nothing (aka. null)!!
```

Instantiating an Object

```
String courseName = new String("CS 121");
     // more on Strings in a minute
Scanner scan = new Scanner(System.in);
```

- When we use the `new` operator, this calls the class constructor – a special method that sets up the object.
- The new object is an instance of the class “blueprint”.

Instantiating String Objects (a special case)

- We don't have to use the new operator to create a String.

```
String courseName = new String("CS 121");
// is the same as
String courseName = "CS 121";
```

- This is *only* supported for String objects (because they are so frequently used). The Java compiler creates the object for us as a convenience.

Reference Variables vs. Primitive Variables

- Primitive variables and object variables store different information.
- Primitive variables (e.g. `int`, `char`, `boolean`) contain the *value* itself.

```java
int courseNumber = 121;
```

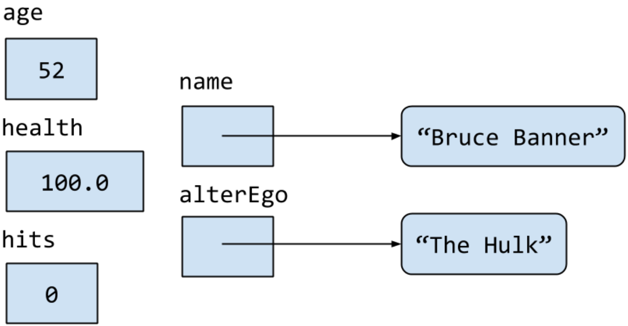
- The variable referring to an object is known as a reference variable.

```java
String courseName = new String("CS 121");
```

- It holds the address (aka. reference) of where the actual object data is stored in memory. We sometimes say it "points to" the object.

Reference Variables: The Hulk

```
int age = 52;
String name = new String(“Bruce Banner”);
String alterEgo = “The Hulk”;
double health = 100.0;
int hits = 0;
```



age

52

health

100.0

hits

0

name

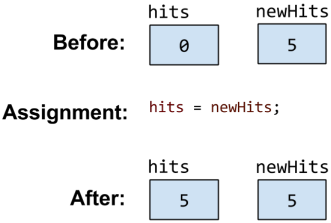
“Bruce Banner”

alterEgo

“The Hulk”

Assignment Revisited

- **Recall:** The act of assignment takes a copy of a value (the *Right-Hand-Side) and stores it in the target variable (the* Left-Hand-Side).
- For primitive types, the *value* of the variable is copied.



Before:

hits	newHits
0	5

Assignment:

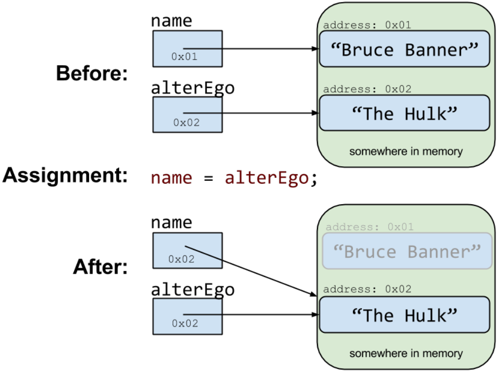
```
hits = newHits;
```

After:

hits	newHits
5	5

Assignment Revisited

- For objects, the *address* of the object is copied.



```
name = alterEgo;
```

Aliases

- Two or more references that refer to the same object are aliases of each other.
- A single object can be accessed using multiple references.
- This is useful, but can cause issues if not managed properly.
- *Changing an object through one reference changes it for all of its aliases, because there is really only one object stored in memory.*

Garbage Collection

- If there are no variables that refer to an object, the object is inaccessible and referred to as garbage.
- Java performs automatic garbage collection in the background, reclaiming the memory used by garbage objects for future use.
- In some languages, the programmer is responsible for freeing the memory used by garbage objects.

Invoking Methods of an Object

- After we instantiate an object, we can use the dot operator to invoke its methods (or behaviors).
- Invoking an object's method can be thought of as asking the object to do something.
- Methods may return values that can be used in an assignment or expression.

```
String courseName = new String("CS 121");

int length = courseName.length();
// the value of length will be 6
```

Invoking Methods of an Object

- Methods may also accept parameters that provide additional information that it may need to perform the requested behavior.

```java
String courseName = new String("CS 121");

String newCourse = courseName.replace('1', '2');
// the value of newCourse will be "CS 222"

char first = courseName.charAt(0);
// the value of first will be 'C'
```

In-Class Exercise

- What is the difference between a class and an object?
- Objects are encapsulated. What does this mean?
- What does it mean to instantiate an object? Write one line of code to instantiate a `Scanner` to read from `System.in`.
- What is a reference variable?
- What is the value of a reference variable if the object it is supposed to refer to is not instantiated?
- How do you tell an object to perform an action/behavior?

Intro to Java Graphics

- The Graphics class from the java.awt package is a useful class for drawing shapes on a canvas.
  - See the Intro to Graphics notes for details on how to use the Graphics class.

The Random Class

- The Random class provides methods that generate pseudorandom numbers. The class is part of the java.util package.
- True random numbers are usually generated from nature or physical processes.
- Give some examples of physical processes that generate random numbers:
  - Flipping a coin
  - Rolling dice
  - Shuffling playing cards
  - Brownian motion of molecules in a liquid
- Pseudorandom numbers are generated using algorithms that start with a seed value. The values generated pass statistical tests. There are two main advantages of pseudorandom numbers:
  - Unlimited supply
  - Reproducibility
- Random numbers are used in simulations, security, testing software, design, games and many other areas.

Selected Methods in the Random class

`Random Random()`	Constructor: creates a new pseudorandom generator
`Random Random(long seed)`	Constructor: with a seed value to be able to reproduce random sequence
`int nextInt(int bound)`	returns a random number over the range 0 to `bound-1`
`int nextInt()`	returns a random number over all possible values of `int`
`double nextDouble()`	returns a `double` random number between 0.0 (inclusive) and 1.0 (exclusive)

Using the Random Class

- Import the class, construct an instance and then use the appropriate methods.

```
import java.util.Random;
Random generator = new Random();
System.out.println(generator.nextInt(10));
System.out.println(generator.nextInt(10));
```

- Use the constructor with a seed argument to create a pseudorandom number sequence that is the same each time:

```
import java.util.Random;
long seed = 12345; //arbitrary number!
Random generator = new Random(seed);
System.out.println(generator.nextInt(10));
System.out.println(generator.nextInt(10));
```

- Example: DiceRoll.java, RandomNumbers.java

In-Class Exercises

1. Given an Random object named generator, what range of values are produced by the following expressions?

   ```
   generator.nextInt(25)
   generator.nextInt(10) + 1
   generator.nextInt(50) + 100
   generator.nextInt(10) - 5
   generator.nextInt(21) - 10
   ```

2. Write an expression using generator that produces the following range of random values:

   0 to 12

   1 to 100

   15 to 20

   -10 to 0

3. Create a random color using the Color class and the Random class.
4. Create a random position within a grid of width, w, and height, h, using the Random class.

Formatting Output (1)

- The java.text package provides several classes to format values for output.
- In this course, we will focus on classes for formatting *numbers*.
  - NumberFormat: formats numerical values (e.g. currency or percentage).
  - DecimalFormat: (a sub-class of NumberFormat) formats decimal values based on a pattern.
- The following code snippets that show the usage of the NumberFormat class.
- The import statement will be at the top of the Java source file.

```java
import java.text.NumberFormat;
```

```java
NumberFormat currFmt = NumberFormat.getCurrencyInstance();
double amount = 1150.45;
System.out.println("Amount: " + currFmt.format(amount));
```

```java
NumberFormat percFmt = NumberFormat.getPercentInstance();
double passRate = .8845;
System.out.println("Amount: " + percFmt.format(passRate));
```

- Example: BasicNumberFormat.java, Purchase.java

Formatting Output (2)

- The DecimalFormat allows us to format values based on a pattern.
  - For example, we can specify the number should be rounded to three digits after the decimal point.
  - Uses Half Even Rounding to truncate digits: round towards the "nearest whole neighbor" unless both whole neighbors are equidistant, in which case, round towards the even neighbor. See here for details: http://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html
- A code snippet that shows the usage:

```java
import java.text.DecimalFormat;
```

```java
DecimalFormat fmt = new DecimalFormat("0.###");
double amount = 110.3424;
System.out.println("Amount: " + fmt.format(amount));
//shows 110.342
```

- We can change the rounding mode with the setRoundingMode method:

```java
fmt.setRoundingMode(RoundingMode.CEILING);
System.out.println("Amount: " + fmt1.format(amount));
//shows 110.343
```

- Example: BasicDecimalFormat.java

Formatting Output (3)

- Commonly used symbols in the pattern:

`0`	digit (`int`, `short`, `byte`)
`#`	digit, zero shows as absent
`.`	decimal separator
`,`	grouping separator (for large numbers)
`E`	show in scientific notation

- Example: CircleStatsDecimalFormat.java
- We can set minimum and maximum limits on integer and fractional digits. For more information, see the javadocs for the `DecimalFormat` class.

In-Class Exercise

What do the following patterns accomplish?

1. `"##.###"`
2. `"00.###"`
3. `"###,###"`
4. `"000,000"`

Formatting Output (4)

- The class `Formatter` from the `java.util` package provides an alternative way of formatting output that is inspired by the `printf` method in C language.

```
import java.util.Formatter;
```

```
Formatter fmt = new Formatter(System.out);
double area = 1150.45;
fmt.format("The area is %f\n", area);
```

- Here the %f is a conversion template that says to format the variable `area` as a floating point number and insert in the output. Various conversions are available for printing a wide variety of types.
- Convenience methods exist in the `System.out` object to use `Formatter` class methods.

```
System.out.printf("The area is %f\n", area);
```

- We can also format a `String` object, which often comes in handy.

```
String output = String.format("The area is %f\n", area);
```

- In each case, the underlying method used is the same.

Selected `printf` Style Formatting Conversions

- Commonly used *conversions*:

`%d`	decimal (`int`, `short`, `byte`)
`%ld`	`long`
`%f`	floating point (`float`, `double`)
`%e`	floating point in scientific notation
`%s`	`String`
`%b`	`boolean`

- Some examples of variations on the default formatting:

`%10d`	use a field 10 wide (right-aligned for numeric types)
`%8.2f`	use a field 8 wide, with two digits after the decimal point
`%-10s`	left justified string in 10 spaces (default is right justified)

- Note that if the output doesn't fit in the number of spaces specified, the space will expand to fit the output.
- Examples: CircleStatsFormatter.java, CircleStatsPrintfTable.java, PrintfExample.java,

The Math Class

- The Math contains methods for basic mathematical operations like exponentiation, square root, logarithm and trigonometric functions.
- Part of the java.lang package so no need to import.
- The methods in the Math class are static methods (also known as class methods).
- Static methods can be invoked using the class name — no Math object needs to be instantiated. For example:

```
double value = Math.sin(Math.PI) + Math.cos(Math.PI);
```

- Examples: Quadratic.java, TrigDemo.java

Selected Methods in the Math class

```
static int abs(int num)
static double sqrt(double num)
static double ceil(double num)
static double floor(double num)

static double log(double num)
static double log10(double num)
static double pow(double num, double power)

static double min(double num1, double num2)
static double max(double num1, double num2)
static int min(int num1, int num2)
static int max(int num1, int num2)

static double sin(double angleInRadians)
static double cos(double angleInRadians)
static double tan(double angleInRadians)
static double toRadians(double angleInDegrees)
static double toDegrees(double angleInRadians)
```

The String Class

- In Java, strings are immutable: Once we create a String object, we cannot change its value or length.
- The String class provides several useful methods for manipulating String objects. Many of these return a new String object since strings are immutable. For example:

  ```
  String babyWord = "googoo";
  String str = babyWord.toUpperCase();
  ```

- See javadocs for String for list of available methods: http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Selected Methods in String class

`int length()`
`char charAt (int index)`

`String toLowerCase()`
`String toUpperCase()`
`String trim()`

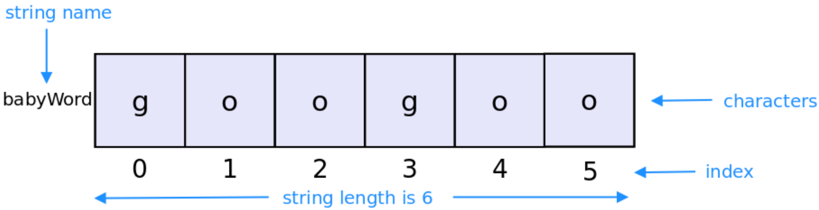
`boolean equals(String str)`
`boolean equalsIgnoreCase(String str)`
`int compareTo(String str)`

`String concat(String str)`
`String replace(char oldChar, char newChar)`
`String substring(int offset, int endIndex)`
returns a string that equals the substring from index `offset` to `endIndex - 1`

`int indexOf(char ch)`
`int indexOf(String str)`
returns the index of the first occurrence of character `ch` or string `str`

String Representation

- The String class represents a string internally as a series of characters. These characters have an index that we can use to refer to a specific character.



- We can use the charAt(int index) method to get the character at the index position.

```
char ch = babyWord.charAt(0);
char ch = babyWord.charAt(4);
```

String Examples

- Example: StringPlay.java
- What output is produced by the following code?

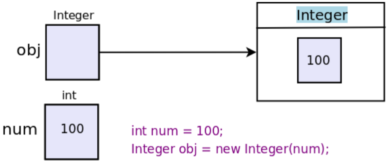
```
String babyWords = "googoo gaagaa";
System.out.println(babyWords.length());
System.out.println(babyWords.toUpperCase());
System.out.println(babyWords.substring(7, 10));
System.out.println(babyWords.replace('g', 'm'));
System.out.println(babyWords.length());
```

Wrapper Classes (1)

- The `java.lang` package contains wrapper classes corresponding to each primitive type.

`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
`void`	`Void`

- See below for the relationship between the wrapper object and the primitive type:



```
int num = 100;
Integer obj = new Integer(num);
```

- An object of a wrapper class can be used any place where we need to store a primitive value as an object.

Wrapper Classes (2)

- The wrapper classes contain useful static methods as well as constants related to the base primitive type.
- For example, the minimum int value is Integer.MIN_VALUE and the maximum int value is Integer.MAX_VALUE.
- Example: PrimitiveTypes.java
- For example, the parseInt method converts an integer stored as a String into an int value. Here is a typical usage to convert input from a user to an integer.

```
Scanner scan = new Scanner(System.in);
String input = scan.nextLine();
int num = Integer.parseInt(input);
```

Wrapper Classes (3)

- Selected methods from the `Integer` class.

`Integer(int value)`

Constructor: builds a new `Integer` object that stores the specified value.

`static parseInt(String s)`

Returns an `int` value corresponding to the value stored in the string `s`.

`static toBinaryString(int i)`
`static toOctalString(int i)`
`static toHexString(int i)`

Returns the string representation of integer `i` in the corresponding base.

- Similar methods and many more are available for all the wrapper classes. Explore the javadocs for the wrapper classes.

Autoboxing

- Autoboxing is the automatic conversion of a primitive value to a corresponding wrapper object.

```
Integer obj;
int num = 100;
obj = num;
```

- The assignment creates the corresponding wrapper `Integer` object. So it is equivalent to the following statement.

```
obj = new Integer(num);
```

- The reverse conversion (unboxing) also happens automatically as needed.

Summary

- Understand the difference between primitive type variables and reference variables.
- Creating and using objects.
- Using `String`, `Math`, `Random`, `Scanner` classes.
- Formatting output using `NumberFormat`, `DecimalFormat` and `Formatter` classes.
- Wrapper classes and autoboxing: `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Character`, `Boolean`

Exercises

- Read Chapter 3.
- **Recommended Homework**:
  - Exercises: EX 3.2, 3.3, 3.4, 3.6, 3.7, 3.11, 3.12.
  - Projects: PP 3.2, 3.3, 3.5.