Building Java Programs

Chapter 4:
Conditional Execution

Lecture 4-2: Objects, `String` Objects

Copyright 2008 by Pearson Education

Objects

reading: 3.3

Copyright 2008 by Pearson Education

Objects and classes

- **object:** An entity that contains:
  - data (variables),
  - behavior (methods).

- **class**: A program, or a template for a type of objects.

- Examples:
  - The class `String` represents objects that store text.
  - The class `DrawingPanel` represents objects that can display drawings.
  - The class `Scanner` represents objects that read information from the keyboard, files, and other sources.

Copyright 2008 by Pearson Education

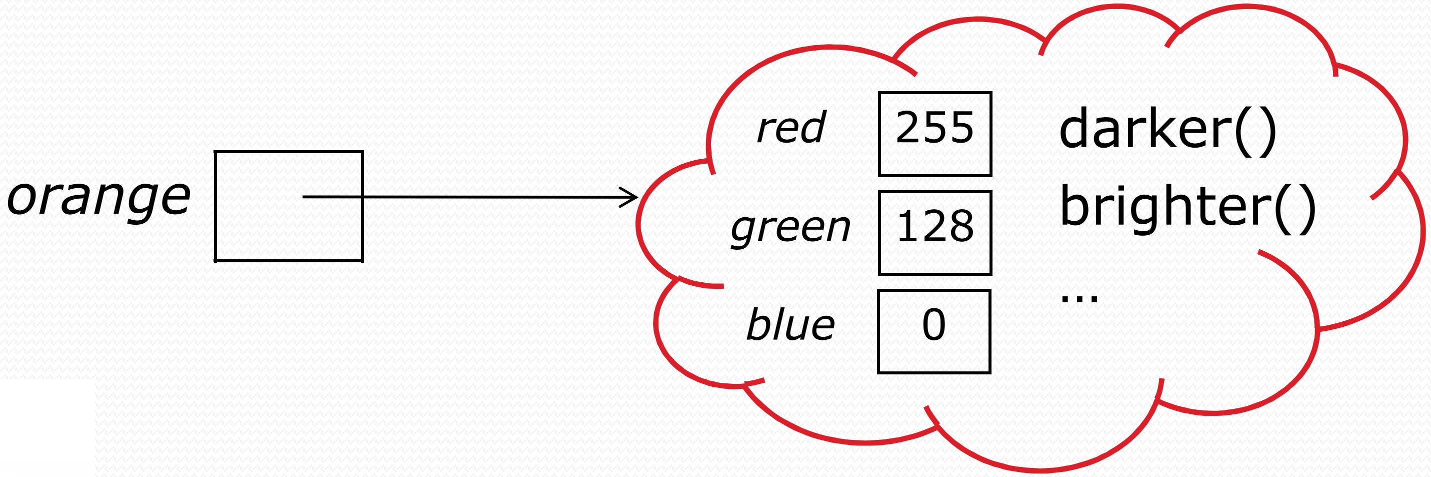
Constructing objects

- Constructing (creating) objects, general syntax:

  <type> ***<name>*** = new ***<type>*** (***<parameters>***);

```
DrawingPanel p = new DrawingPanel(300, 200);
Color orange = new Color(255, 128, 0);
```

  - The variable contains an address to find the object in memory



Copyright 2008 by Pearson Education

Calling methods of objects

- Objects have methods that your program can call.
  - The methods often relate to the data inside the object.

- Syntax:

  <object> . ***<method name>*** (***<parameters>***)

  - Examples:

```
DrawingPanel p = new DrawingPanel(100, 100);
Color orange = new Color(255, 128, 0);
p.setBackground(orange.darker());
```

Copyright 2008 by Pearson Education

Value and reference semantics

reading: 3.3, 4.3

Copyright 2008 by Pearson Education

Swapping values

```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b (incorrectly)
    a = b;
    b = a;

    System.out.println(a + " " + b);
}
```

- What is wrong with this code? What is its output?

- The red code should be replaced with:

```
    int temp = a;
    a = b;
    b = temp;
```

Copyright 2008 by Pearson Education

A swap method?

- The following swap method does not work? Why not?

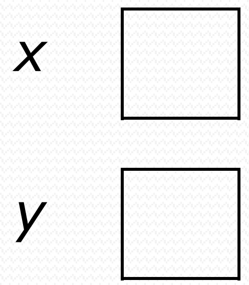
```
public static void main(String[] args) {
    int a = 7;
    int b = 35;

    // swap a with b
    swap(a, b);

    System.out.println(a + " " + b);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}
```

Copyright 2008 by Pearson Education

Value semantics

- **value semantics**: Behavior where variables are copied when assigned to each other or passed as parameters.
  - One primitive variable assigned to another gets a copy of the value.
  - Modifying the value of one variable does not affect others.

```
int x = 5;
int y = x;      // x = 5, y = 5
x = 8;          // x = 8, y = 5
y = 17;         // x = 8, y = 17
```

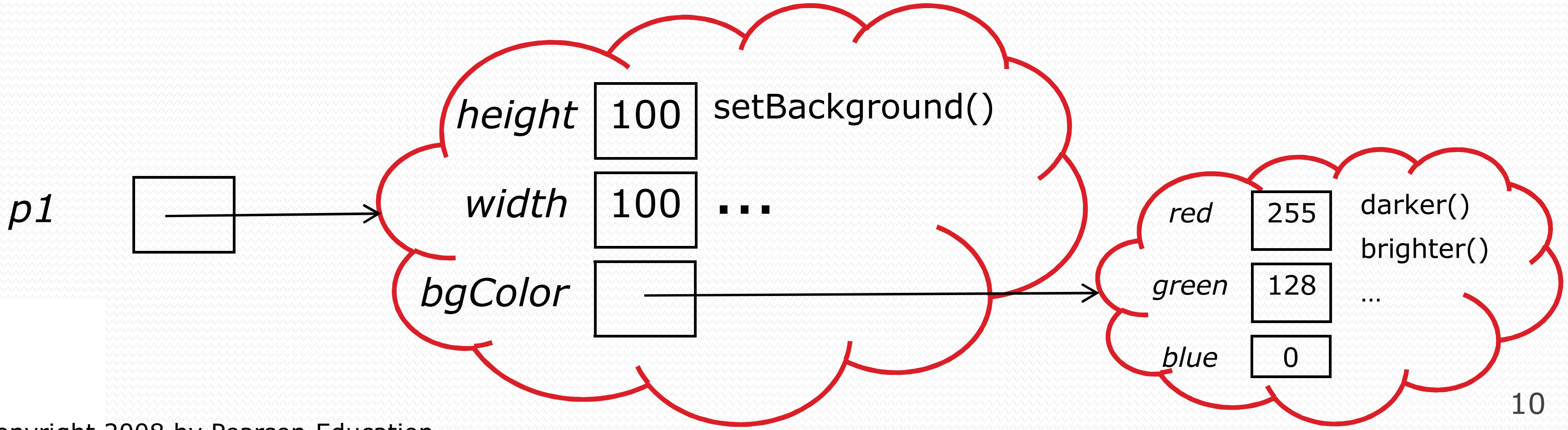


Copyright 2008 by Pearson Education

Reference semantics

- **reference semantics**: Behavior where multiple variables can refer to a common value (object).
  - *Reference variables* store an object's address in memory.
- Why is it done this way?
  - *efficiency.* Copying large objects slows down a program.
  - *sharing.* It's useful to share an object's data among methods.

```
DrawingPanel p1 = new DrawingPanel(100, 100);
```

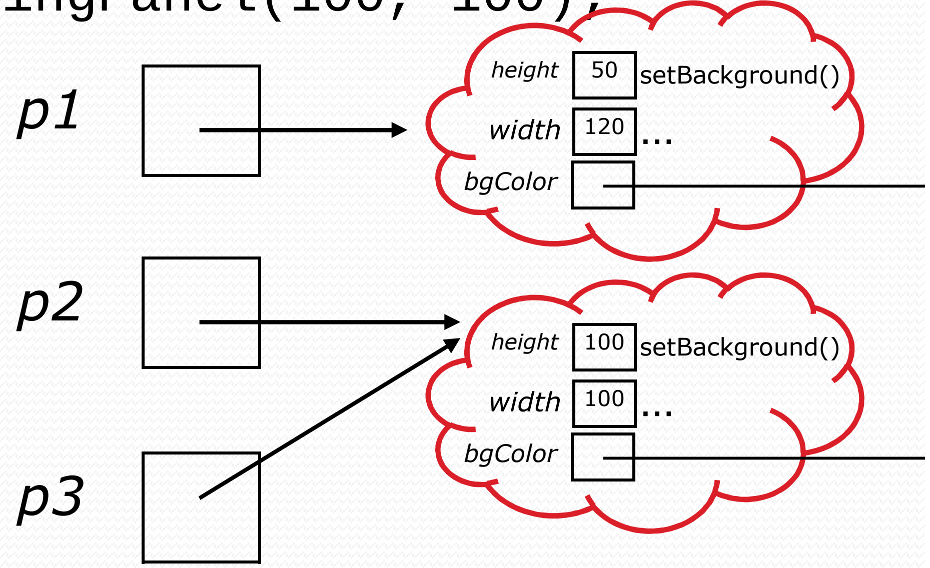


Copyright 2008 by Pearson Education

Multiple references

- If one reference variable is assigned to another, the object is *not* copied. The variables share the object.
  - Calling methods on either variable modifies the same object.

```
DrawingPanel p1 = new DrawingPanel(120, 50);
DrawingPanel p2 = new DrawingPanel(100, 100);
DrawingPanel p3 = p2;
// No new panel pops up

p3.setBackground(orange);
// Changes color of
// single 100x100 panel
```



Copyright 2008 by Pearson Education

Objects as parameters

- When objects are passed, they are shared, not copied.
  - You can pass an object to a method, let the method change its data, and the caller will also see that change.

```
public static void main(String[] args) {
    DrawingPanel p = new DrawingPanel(100,100);
    Graphics gr = p.getGraphics();
    example1(gr);
    example2(gr);
}

public static void example1(Graphics g) {
    g.drawRect(10,10,10,10);
}

public static void example2(Graphics g) {
    g.drawRect(80,80,10,10);
}
```

Copyright 2008 by Pearson Education

`String` objects

reading: 3.3, 4.4

self-check: Chap. 4 #12, 15
exercises: Chap. 4 #15, 16

Copyright 2008 by Pearson Education

Strings

- **String**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

```
String <name> = "<text>";
String <name> = <expression>;
```

  - Examples:

```
String name = "Marla Singer";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Copyright 2008 by Pearson Education

Indexes

- The characters are numbered with 0-based *indexes*:

```
String name = "P. Diddy";
```

name

index	0	1	2	3	4	5	6	7
char	`P`	`.`		`D`	`i`	`d`	`d`	`y`

  - The individual characters are values of type `char` (seen later)

Copyright 2008 by Pearson Education

String methods

Method name	Description
`indexOf(`***str***`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`***index1***`, `***index2***`)`
or
`substring(`***index1***`)`
	the characters in this string from ***index1*** (inclusive) to ***index2*** (exclusive);
	if ***index2*** omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String message = "and Dr. Dre said";
System.out.println(message.length());   // 16
```

Copyright 2008 by Pearson Education

String method examples

```
//      index 012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());              // 12
System.out.println(s1.indexOf("e"));          // 8
System.out.println(s1.substring(7, 10));      // Reg

String s3 = s2.substring(3, 8);
System.out.println(s3.toLowerCase());         // ty st
```

- Given the following string:

```
//                0123456789012345678901
String book = "Building Java Programs";
```

  - How would you extract the word "Java" ?
  - Change book to store "BUILDING JAVA PROGRAMS" .
  - How would you extract the first word from any string?

Copyright 2008 by Pearson Education

Modifying strings

- Methods like `substring`, `toLowerCase`, `toUpperCase`, etc. actually create and return a new string:

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);   // lil bow wow
```

- To modify the variable, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);   // LIL BOW WOW
```

Copyright 2008 by Pearson Education

Comparing objects

- Relational operators such as < and == fail on objects.
  - The == operator on `String`s often evaluates to `false` even when two `String`s have the same letters.
  - Example (*bad code*):

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - This code will compile, but it will never print the song.

Copyright 2008 by Pearson Education

The equals method

- Objects (e.g. `String`, `Color`) should be compared using a method named `equals`.
  - Example:

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

Copyright 2008 by Pearson Education

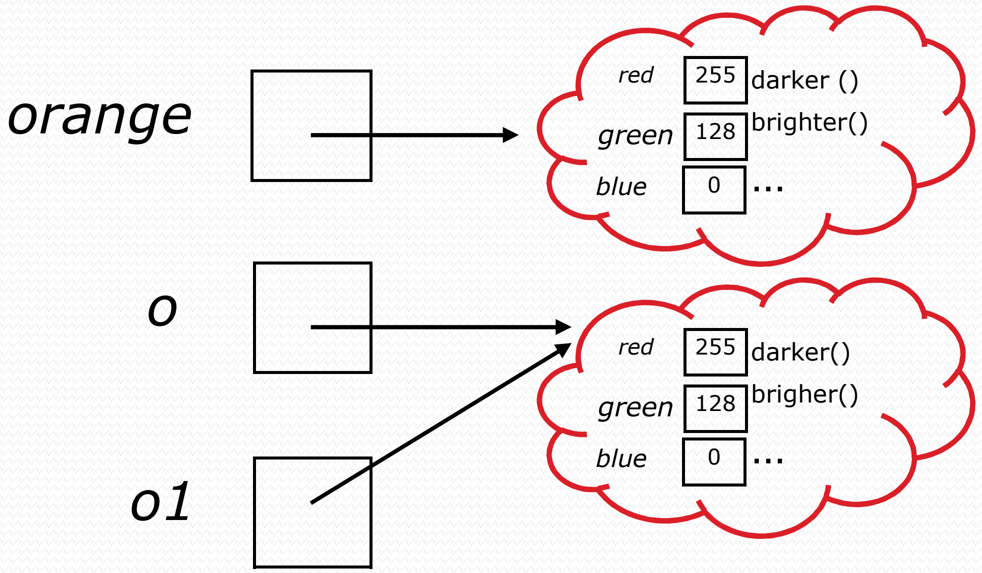
== vs. equals

- == compares whether two variables refer to the same object.
- equals compares whether two objects have the same state.
  - Given the following code:

```
Color orange = new Color(255, 128, 0);
Color o = new Color(255, 128, 0);
Color o1 = o;
```

  - Which tests are true?

```
orange == o
orange == o1
o == o1
orange.equals(o)
orange.equals(o1)
o.equals(o1)
```



Copyright 2008 by Pearson Education

String test methods

Method	**Description**
`equals(`***str***`)`	whether two strings contain the same characters
`equalsIgnoreCase(`***str***`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`***str***`)`	whether one contains other's characters at start
`endsWith(`***str***`)`	whether one contains other's characters at end

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Is he single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
    System.out.println("I need your TPS reports.");
}
```

Copyright 2008 by Pearson Education

Strings question

- Write a program that judges a couplet by giving it one point if it
  - is composed of two verses with lengths within 4 chars of each other,
  - *"rhymes"* (the two verses end with the same last two letters),
  - *alliterates* (the two verses begin with the same letter).
- A couplet which gets 2 or more points is "good"

Example logs of execution:

(run #1)

```
First verse: I joined the CS party
Second verse: Like "LN" and Marty
2 points: Keep it up, lyrical genius!
```

(run #2)

```
First verse: And it's still about the Benjamins
Second verse: Big faced hundreds and whatever other synonyms
0 points: Aw, come on.  You can do better...
```

Copyright 2008 by Pearson Education

Strings answer

```
// Determines whether a two-verse lyric is "good."
import java.util.*;

public class CheckCouplet {
    public static void main(String[] args) {
        System.out.println("Let's check that couplet!\n");
        Scanner console = new Scanner(System.in);
        System.out.print("First verse: ");
        String verse1 = console.nextLine().toLowerCase();
        System.out.print("Second verse: ");
        String verse2 = console.nextLine().toLowerCase();
        int points = 0;

        // check lengths
        if(Math.abs(verse1.length() - verse2.length()) <= 4) {
            points++;
        }

        // check whether they end with the same two letters
        if(verse2.length() >= 2 &&
           verse1.endsWith(verse2.substring(verse2.length() - 2)));
            points++;
        }

        // check whether they alliterate
        if(verse1.startsWith(verse2.substring(0, 1))) {
            points++;
        }
    }
}
```

Copyright 2008 by Pearson Education