

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is partially visible, extending from the left edge towards the center of the frame.

Building Java Programs

Chapters 3-4: Using Objects

Chapter outline

- objects
 - Point objects
 - String objects
- value vs. reference semantics
 - comparing objects

A brick wall is visible on the left side of the slide, extending from the bottom to the top. The bricks are reddish-brown with white mortar. The background is a solid blue color.

Using objects

reading: 3.3

Objects and classes

- **object:** An entity that contains data and behavior.
 - Variables inside the object store its data.
 - Methods inside the object implement its behavior.
- **class:** A program, or a type of objects.
 - Classes' names are uppercase (e.g. Point, Color).
- **Examples:**
 - Scanner objects read data from the keyboard and other sources.
 - DrawingPanel objects represent graphical windows.
 - What data and behavior do these objects have?

Constructing objects

- Constructing (creating) objects, general syntax:

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

- Examples:

```
Scanner console = new Scanner(System.in);
```

```
DrawingPanel window = new DrawingPanel(300, 200);
```

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

```
Point p = new Point(7, -4);
```

Calling methods of objects

- Objects have methods that your program can call.
 - The methods often relate to the data inside the object.
- Calling an object's method, general syntax:

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

- Examples:

```
Scanner console = new Scanner(System.in);  
int age = console.nextInt();
```

```
Point p1 = new Point(3, 4);  
Point p2 = new Point(0, 0);  
System.out.println(p1.distance(p2));           // 5.0
```

Point objects

- Java has a class of objects named `Point`.
 - They store two values, an (x, y) pair, in a single variable.
 - They have useful methods we can call in our programs.
 - To use `Point`, you must write:

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

- Two ways to construct a `Point` object:

```
Point <name> = new Point(<x>, <y>);
```

```
Point <name> = new Point(); // the origin (0, 0)
```

- Examples:

```
Point p1 = new Point(5, -2);
```

```
Point p2 = new Point();
```

Point data and methods

- Data stored in each `Point` object:

Field name	Description
<code>x</code>	the point's x-coordinate
<code>y</code>	the point's y-coordinate

- Methods of each `Point` object:

Method name	Description
<code>distance(<i>p</i>)</code>	how far away the point is from point <i>p</i>
<code>setLocation(<i>x</i>, <i>y</i>)</code>	sets the point's x and y to the given values
<code>translate(<i>dx</i>, <i>dy</i>)</code>	adjusts the point's x and y by the given amounts

- Point objects can also be printed using `println` statements:

```
Point p = new Point(5, -2);  
System.out.println(p);    // java.awt.Point[x=5,y=-2]
```


Point example

- Write a program that computes a right triangle's perimeter, given two of its side lengths a and b . (It's the sum of sides $a+b+c$)

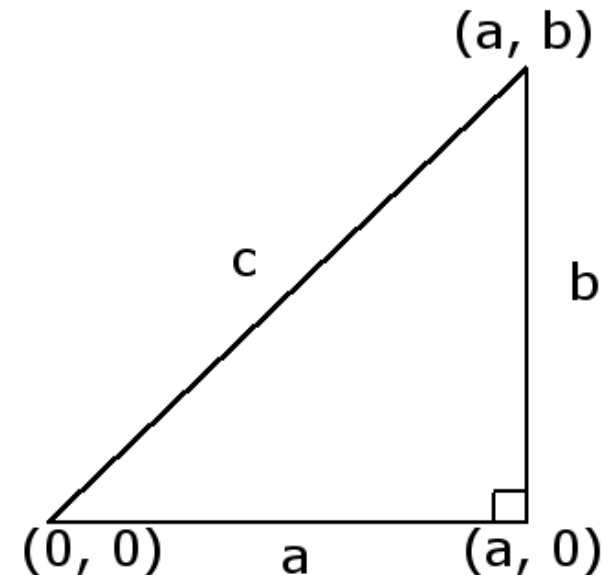
```
import java.awt.*;    // for Point
import java.util.*;  // for Scanner

public class TrianglePerimeter {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("side a? ");
        int a = console.nextInt();
        System.out.print("side b? ");
        int b = console.nextInt();

        // finish me
    }
}
```

Example Output:

```
side a? 12
side b? 5
perimeter is 30.0
```



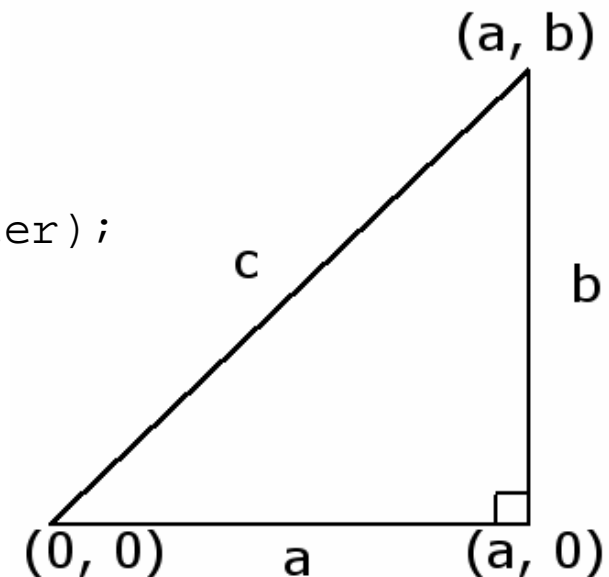
Point example answer

- Computing a right triangle's perimeter (sum of sides $a+b+c$):

```
import java.awt.*;    // for Point
import java.util.*;  // for Scanner

public class TrianglePerimeter {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("side a? ");
        int a = console.nextInt();
        System.out.print("side b? ");
        int b = console.nextInt();

        Point p1 = new Point();           // 0, 0
        Point p2 = new Point(a, b);
        double c = p1.distance(p2);
        double perimeter = a + b + c;
        System.out.println("perimeter is " + perimeter);
    }
}
```



A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar. The wall is partially visible, extending from the left edge towards the center of the frame.

Value and reference semantics

reading: 3.3, 4.3

Swapping primitive values

- Consider the following code to swap two `int` variables:

```
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;
```

A swap method?

- We might want to make swapping into a method.
 - Does the following `swap` method work? Why or 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;  
}
```

Value semantics

- **value semantics:** Behavior where variables are copied when assigned to each other or passed as parameters.
 - When one primitive variable is assigned to another, its value is copied.
 - Modifying the value of one variable does not affect others.

```
int x = 5;
```

```
int y = x;
```

```
y = 17;
```

```
x = 8;
```

```
// x = 5, y = 5
```

```
// x = 5, y = 17
```

```
// x = 8, y = 17
```

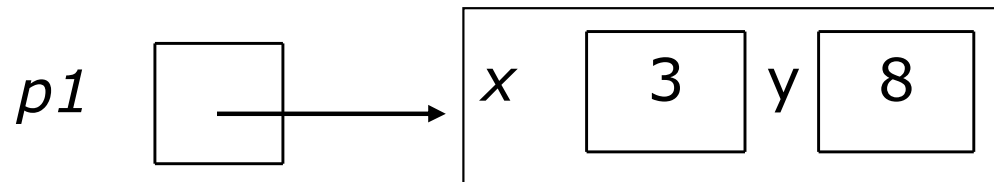
x

y

Reference semantics

- **reference semantics:** Behavior where multiple variables can refer to a common value (object).
 - Variables that store objects are called *reference variables*.
 - Reference variables store the address of an object in memory.

```
Point p1 = new Point(3, 8);
```



- Why is it done this way?
 - *efficiency*. Copying large objects would slow down the program.
 - *sharing*. It's useful to share an object's data between methods.

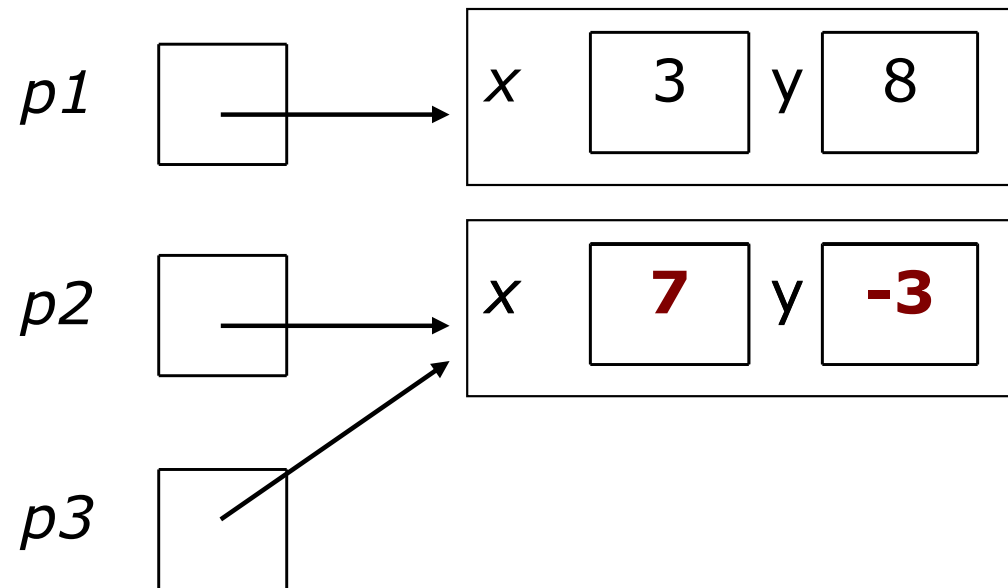
Multiple references

- If one reference variable is assigned to another, the object is *not* copied. The variables share the object.
 - Calling a method on either variable will modify the same object.

```
Point p1 = new Point(3, 8);  
Point p2 = new Point(2, -4);  
Point p3 = p2;
```

```
p3.translate(5, 1);  
System.out.println(p2);
```

```
// OUTPUT:  
// java.awt.Point[x=7,y=-3]
```

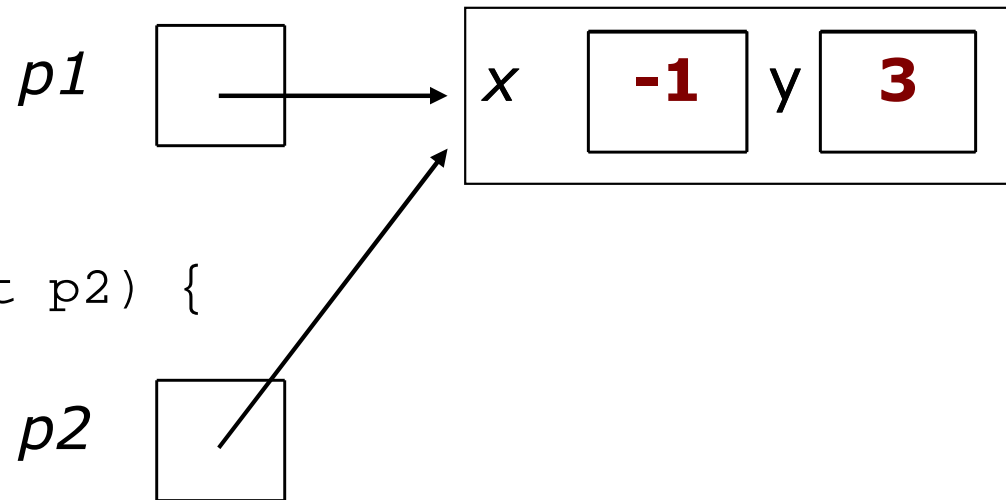


Objects as parameters

- When an object is passed as a parameter, it is not copied. The same object is shared with the parameter.

```
public static void main(String[] args) {  
    Point p1 = new Point(3, 7);  
    example(p1);  
    System.out.println(p1);  
}
```

```
public static void example(Point p2) {  
    p2.setLocation(-1, 3);  
}
```



- This is useful because we can pass an object to a method, let the method change its data, and we will also see that change.

A brick wall on the left side of a blue background. The bricks are reddish-brown with white mortar lines. The wall is partially visible, extending from the left edge towards the center of the frame.

String objects

reading: 3.3

String objects

- **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";
```

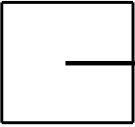
```
int x = 3, y = 5;
```

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

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)

String methods

- Useful methods of each `String` object:

Method name	Description
<code>indexOf(str)</code>	index where the start of the given string appears in this string (-1 if it is not there)
<code>length()</code>	number of characters in this string
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	the characters in this string from index1 (inclusive) to index2 (exclusive); if index2 omitted, grabs till end of string
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String example = "speak friend and enter";  
System.out.println(example.length());
```

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 general string?

String condition methods

- These `String` methods can be used as `if` conditions:

Method	Description
<code>equals(<i>str</i>)</code>	whether two strings contain exactly the same characters
<code>equalsIgnoreCase(<i>str</i>)</code>	whether two strings contain the same characters, ignoring upper vs. lower case
<code>startsWith(<i>str</i>)</code>	whether one string contains the other's characters at its start
<code>endsWith(<i>str</i>)</code>	whether one string contains the other's characters at its 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.");
}
```

Strings question

- Write a program that compares two words typed by the user to see whether they *rhyme* (end with the same last two letters) and/or *alliterate* (begin with the same letter).
 - Example logs of execution:
 - (run #1)
Type two words: car STAR
They rhyme!
 - (run #2)
Type two words: bare bear
They alliterate!
 - (run #3)
Type two words: sell shell
They alliterate!
They rhyme!

Strings answer

```
// Determines whether two words rhyme and/or alliterate.
import java.util.*;

public class Rhyme {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type two words: ");
        String word1 = console.next().toLowerCase();
        String word2 = console.next().toLowerCase();

        // check whether they end with the same two letters
        if (word2.length() >= 2 &&
            word1.endsWith(word2.substring(word2.length() - 2))) {
            System.out.println("They rhyme!");
        }

        // check whether they alliterate
        if (word1.startsWith(word2.substring(0, 1)) {
            System.out.println("They alliterate!");
        }
    }
}
```

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
```

Comparing objects

- Relational operators such as `<` and `==` fail on objects.
 - The `==` operator on `Strings` often evaluates to `false` even when two `Strings` 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.

The equals method

- Objects (e.g. `String`, `Point`, `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!");
}
```

== vs. equals

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

- Given the following code:

```
Point p1 = new Point(3, 8);  
Point p2 = new Point(3, 8);  
Point p3 = p2;
```

- Which tests are true?

```
p1 == p2  
p1 == p3  
p2 == p3  
p1.equals(p2)  
p1.equals(p3)  
p2.equals(p3)
```

