Chapter 8:
A Second Look at Classes and Objects

Starting Out with Java:
From Control Structures through Objects

Fifth Edition

by Tony Gaddis

PEARSON ALWAYS LEARNING

Troy University Resources

Chapter Topics

Chapter 8 discusses the following main topics:

- Static Class Members
- Passing Objects as Arguments to Methods
- Returning Objects from Methods
- The `toString` method
- Writing an `equals` Method
- Methods that Copy Objects

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Chapter Topics

Chapter 8 discusses the following main topics:

- Aggregation
- The `this` Reference Variable
- Enumerated Types
- Garbage Collection
- Focus on Object-Oriented Design: Class Collaboration

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Review of Instance Fields and Methods

- Each instance of a class has its own copy of instance variables.
  - Example:
    - The `Rectangle` class defines a `length` and a `width` field.
    - Each instance of the `Rectangle` class can have different values stored in its `length` and `width` fields.
- Instance methods require that an instance of a class be created in order to be used.
- Instance methods typically interact with instance fields or calculate values based on those fields.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Static Class Members

- *Static fields* and *static methods* do not belong to a single instance of a class.
- To invoke a static method or use a static field, the class name, rather than the instance name, is used.
- Example:

```
double val = Math.sqrt(25.0);
```



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Static Fields

- Class fields are declared using the `static` keyword between the access specifier and the field type.

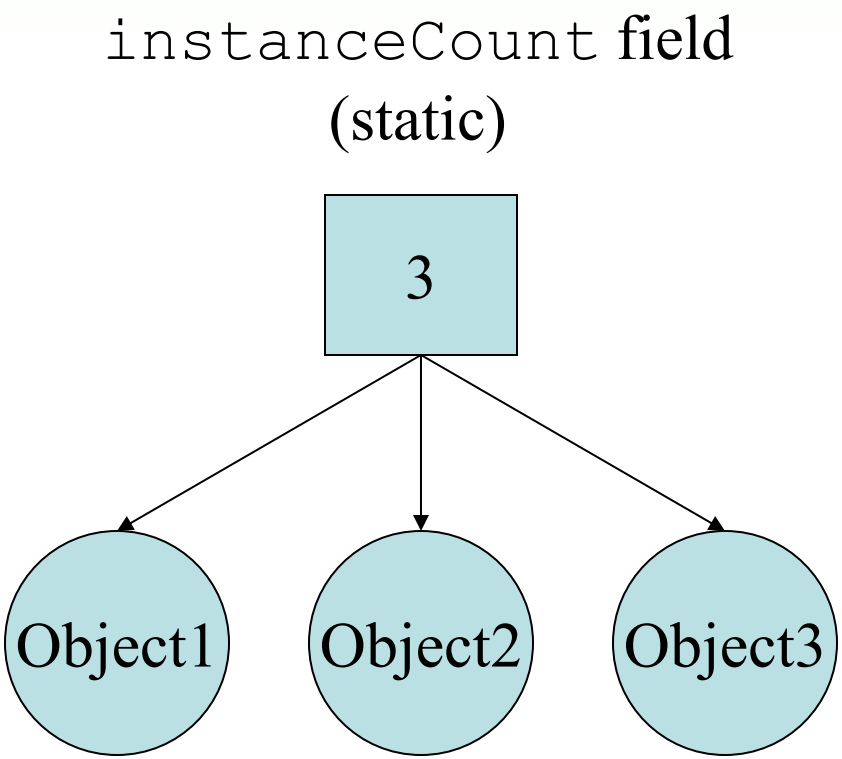
```
private static int instanceCount = 0;
```

- The field is initialized to 0 only once, regardless of the number of times the class is instantiated.
  - Primitive static fields are initialized to 0 if no initialization is performed.

- Examples: Countable.java, StaticDemo.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Static Fields



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Static Methods

- Methods can also be declared static by placing the `static` keyword between the access modifier and the return type of the method.

```
public static double milesToKilometers(double miles)
{…}
```

- When a class contains a static method, it is not necessary to create an instance of the class in order to use the method.

```
double kilosPerMile = Metric.milesToKilometers(1.0);
```

- Examples: Metric.java, MetricDemo.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Static Methods

- Static methods are convenient because they may be called at the class level.
- They are typically used to create utility classes, such as the `Math` class in the Java Standard Library.
- Static methods may not communicate with instance fields, only static fields.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Passing Objects as Arguments

- Objects can be passed to methods as arguments.
- Java passes all arguments *by value*.
- When an object is passed as an argument, the value of the reference variable is passed.
- The value of the reference variable is an address or reference to the object in memory.
- A *copy* of the object is *not passed*, just a pointer to the object.
- When a method receives a reference variable as an argument, it is possible for the method to modify the contents of the object referenced by the variable.

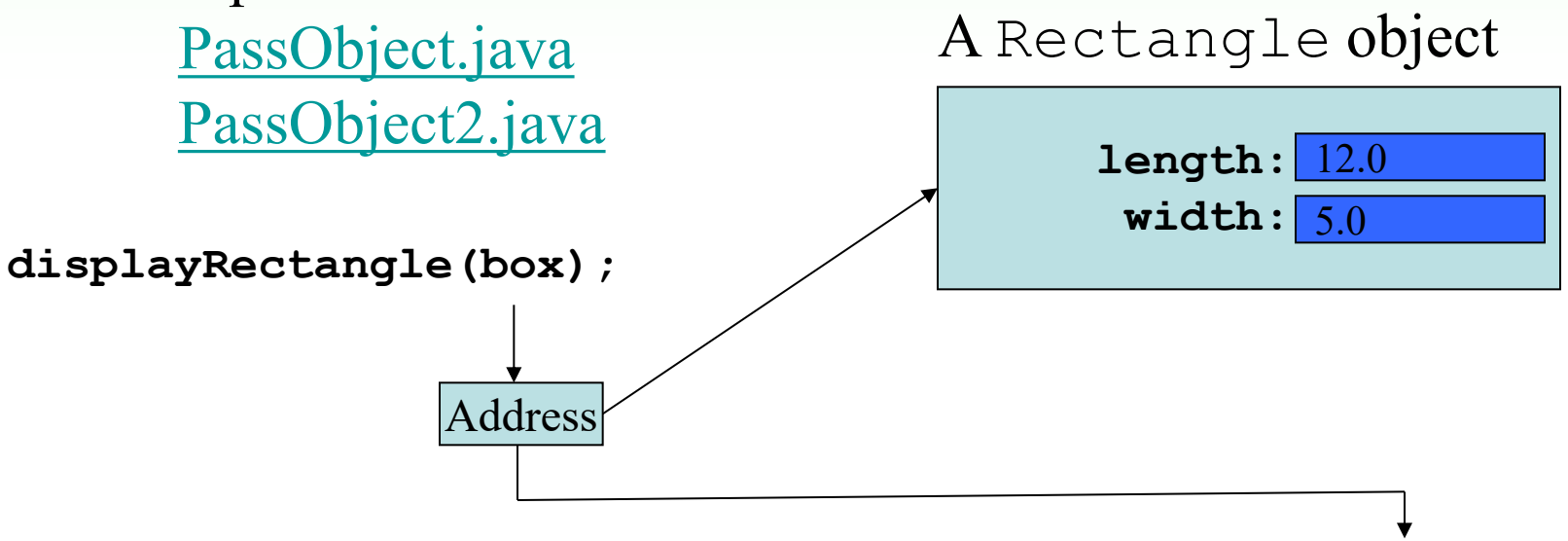
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Passing Objects as Arguments

Examples:

PassObject.java

PassObject2.java




```
public static void displayRectangle(Rectangle r)
{
  // Display the length and width.
  System.out.println("Length: " + r.getLength() +
                     " Width: " + r.getWidth());
}
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Returning Objects From Methods

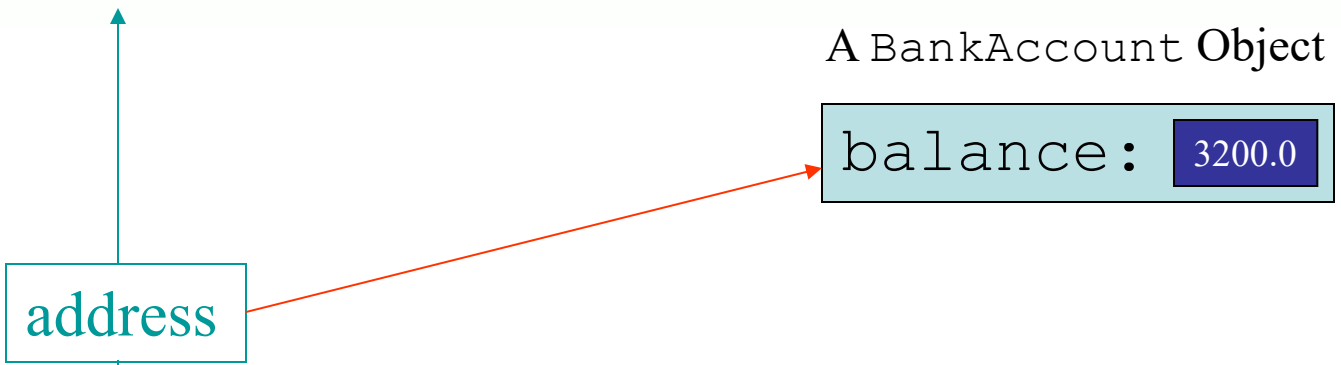
- Methods are not limited to returning the primitive data types.
- Methods can return references to objects as well.
- Just as with passing arguments, a copy of the object is **not** returned, only its address.
- See example: ReturnObject.java
- Method return type:

```
public static BankAccount getAccount()
{
    …
     return new BankAccount(balance);
}
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Returning Objects from Methods

```
account = getAccount();
```




```
public static BankAccount getAccount()
{
    …
    return new BankAccount(balance);
}
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `toString` Method

- The `toString` method of a class can be called *explicitly*:

```
Stock xyzCompany = new Stock ("XYZ", 9.62);
System.out.println(xyzCompany.toString());
```

- However, the `toString` method does not have to be called explicitly but is called implicitly whenever you pass an object of the class to `println` or `print`.

```
Stock xyzCompany = new Stock ("XYZ", 9.62);
System.out.println(xyzCompany);
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `toString` method

- The `toString` method is also called implicitly whenever you concatenate an object of the class with a string.

```
Stock xyzCompany = new Stock ("XYZ", 9.62);
System.out.println("The stock data is:\n" +
   xyzCompany);
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `toString` Method

- All objects have a `toString` method that returns the class name and a hash of the memory address of the object.
- We can override the default method with our own to print out more useful information.
- Examples: Stock.java, StockDemo1.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `equals` Method

- When the == operator is used with reference variables, the memory address of the objects are compared.
- The contents of the objects are not compared.
- All objects have an `equals` method.
- The default operation of the `equals` method is to compare memory addresses of the objects (just like the == operator).

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `equals` Method

- The `Stock` class has an `equals` method.
- If we try the following:

```
Stock stock1 = new Stock("GMX", 55.3);
Stock stock2 = new Stock("GMX", 55.3);
if (stock1 == stock2) // This is a mistake.
  System.out.println("The objects are the same.");
else
  System.out.println("The objects are not the same.");
```

only the addresses of the objects are compared.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `equals` Method

- Instead of using the == operator to compare two `Stock` objects, we should use the `equals` method.

```
public boolean equals(Stock object2)
{
   boolean status;

   if(symbol.equals(Object2.symbol && sharePrice == Object2.sharePrice)
      status = true;
   else
      status = false;
   return status;
}
```

- Now, objects can be compared by their contents rather than by their memory addresses.
- See example: StockCompare.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Methods That Copy Objects

- There are two ways to copy an object.
  - You cannot use the assignment operator to copy reference types
  - Reference only copy
    - This is simply copying the address of an object into another reference variable.
  - Deep copy (correct)
    - This involves creating a new instance of the class and copying the values from one object into the new object.
  - Example: ObjectCopy.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Copy Constructors

- A copy constructor accepts an existing object of the same class and clones it

```
public Stock(Stock object 2)
{
     symbol = object2.symbol;
     sharePrice = object2.sharePrice;
}

// Create a Stock object
Stock company1 = new Stock("XYZ", 9.62);

//Create company2, a copy of company1
Stock company2 = new Stock(company1);
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

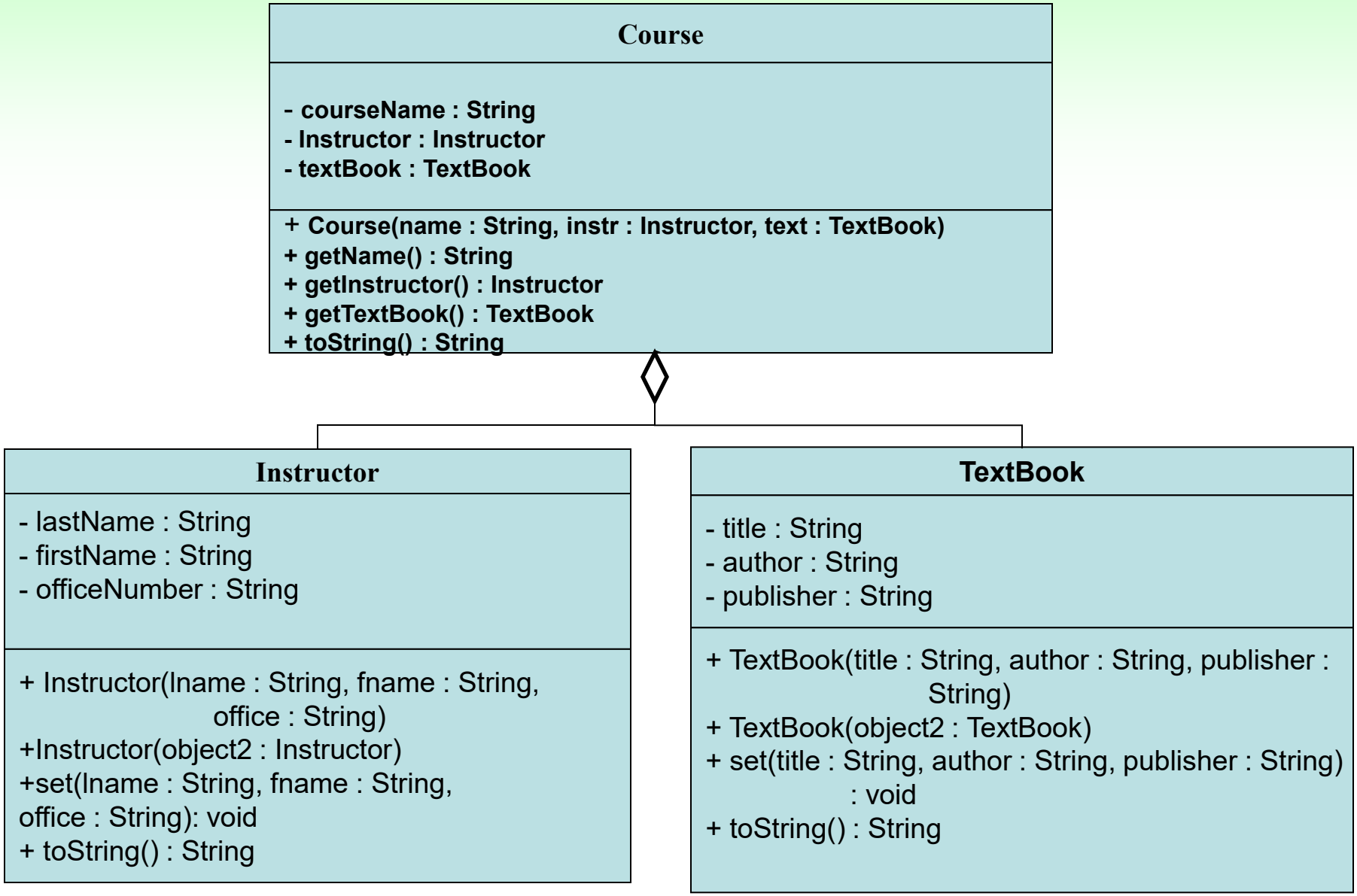
Troy University Resources

Aggregation

- Creating an instance of one class as a reference in another class is called *object aggregation*.
- Aggregation creates a "has a" relationship between objects.
- Examples:
  - Instructor.java, Textbook.java, Course.java, CourseDemo.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Aggregation in UML Diagrams



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Returning References to Private Fields

- Avoid returning references to private data elements.
- Returning references to private variables will allow any object that receives the reference to modify the variable.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Null References

- A *null reference* is a reference variable that points to nothing.
- If a reference is null, then no operations can be performed on it.
- References can be tested to see if they point to null prior to being used.

```
if(name != null)
{
   System.out.println("Name is: "
                      + name.toUpperCase());
}
```

- Examples: FullName.java, NameTester.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `this` Reference

- The `this` reference is simply a name that an object can use to refer to itself.
- The `this` reference can be used to overcome shadowing and allow a parameter to have the same name as an instance field.

```
public void setFeet(int feet)
{
   this.feet = feet;
   //sets the this instance's feet field
   //equal to the parameter feet.
}
```

Local parameter variable feet

Shadowed instance variable

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `this` Reference

- The `this` reference can be used to call a constructor from another constructor.

```
public Stock(String sym)
{
   this(sym, 0.0);
}
```

  - This constructor would allow an instance of the `Stock` class to be created using only the symbol name as a parameter.
  - It calls the constructor that takes the symbol and the price, using *sym* as the symbol argument and 0 as the price argument.
- Elaborate constructor chaining can be created using this technique.
- If `this` is used in a constructor, it must be the first statement in the constructor.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Enumerated Types

- Known as an `enum`, requires declaration and definition like a class
- Syntax:
  `enum` *`typeName`* `{` *one or more enum constants* `}`
  - Definition:

```
enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
           FRIDAY, SATURDAY }
```

  - Declaration:

```
Day WorkDay; // creates a Day enum
```

  - Assignment:

```
Day WorkDay = Day.WEDNESDAY;
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

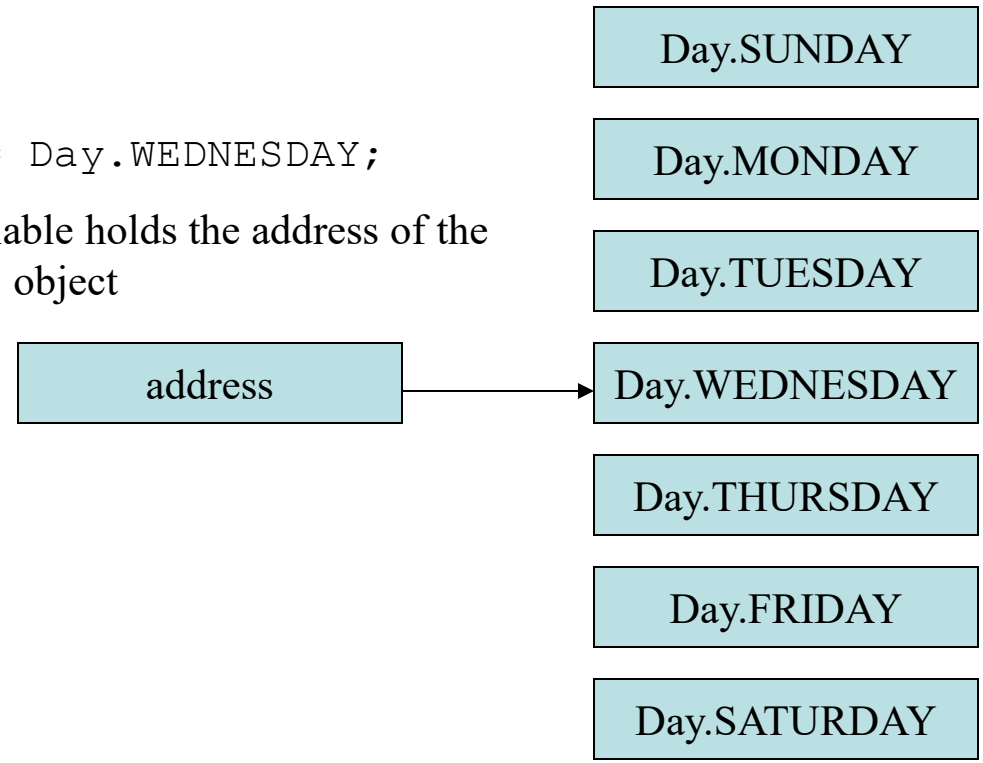
Enumerated Types

- An `enum` is a specialized class

Each are objects of type `Day`, a specialized class

```
Day workDay = Day.WEDNESDAY;
```

The `workDay` variable holds the address of the `Day.WEDNESDAY` object



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Enumerated Types - Methods

- `toString` – returns name of calling constant
- `ordinal` – returns the zero-based position of the constant in the enum. For example the ordinal for `Day.THURSDAY` is 4
- `equals` – accepts an object as an argument and returns true if the argument is equal to the calling enum constant
- `compareTo` - accepts an object as an argument and returns a negative integer if the calling constant's ordinal < than the argument's ordinal, a positive integer if the calling constant's ordinal > than the argument's ordinal and zero if the calling constant's ordinal == the argument's ordinal.
- Examples: EnumDemo.java, CarType.java, SportsCar.java, SportsCarDemo.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Enumerated Types - Switching

- Java allows you to test an enum constant with a `switch` statement.

Example: SportsCarDemo2.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Garbage Collection

- When objects are no longer needed they should be destroyed.
- This frees up the memory that they consumed.
- Java handles all of the memory operations for you.
- Simply set the reference to *null* and Java will reclaim the memory.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Garbage Collection

- The Java Virtual Machine has a process that runs in the background that reclaims memory from released objects.
- The *garbage collector* will reclaim memory from any object that no longer has a valid reference pointing to it.

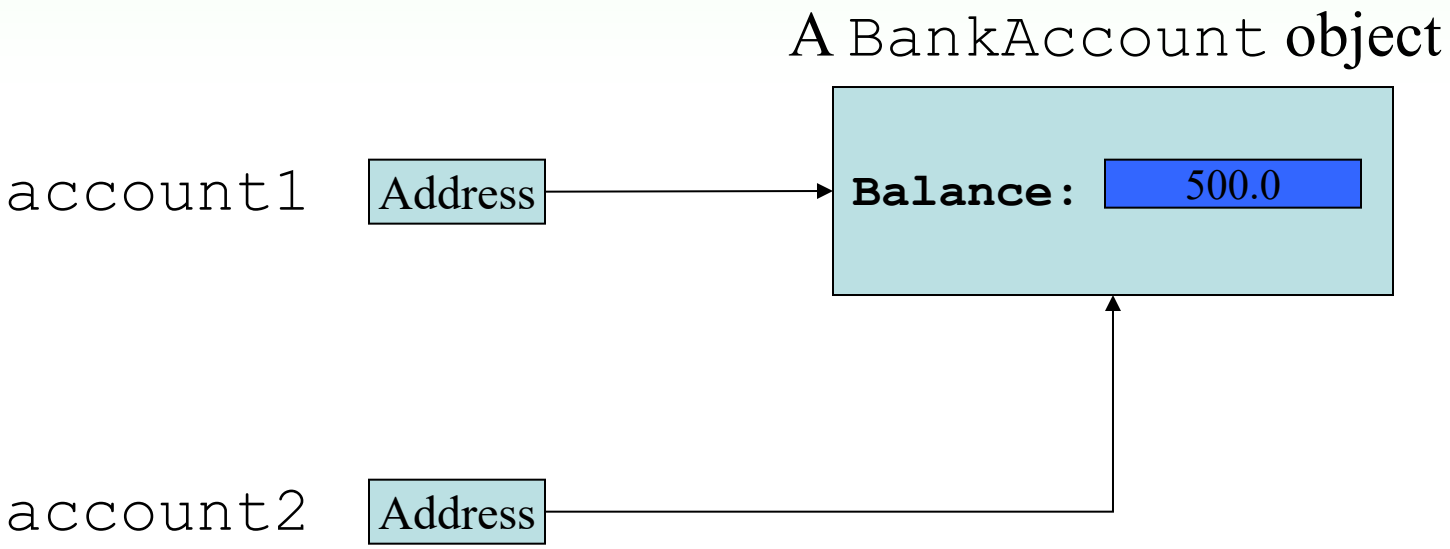
```
BankAccount account1 = new BankAccount(500.0);
BankAccount account2 = account1;
```

- This sets `account1` and `account2` to point to the same object.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

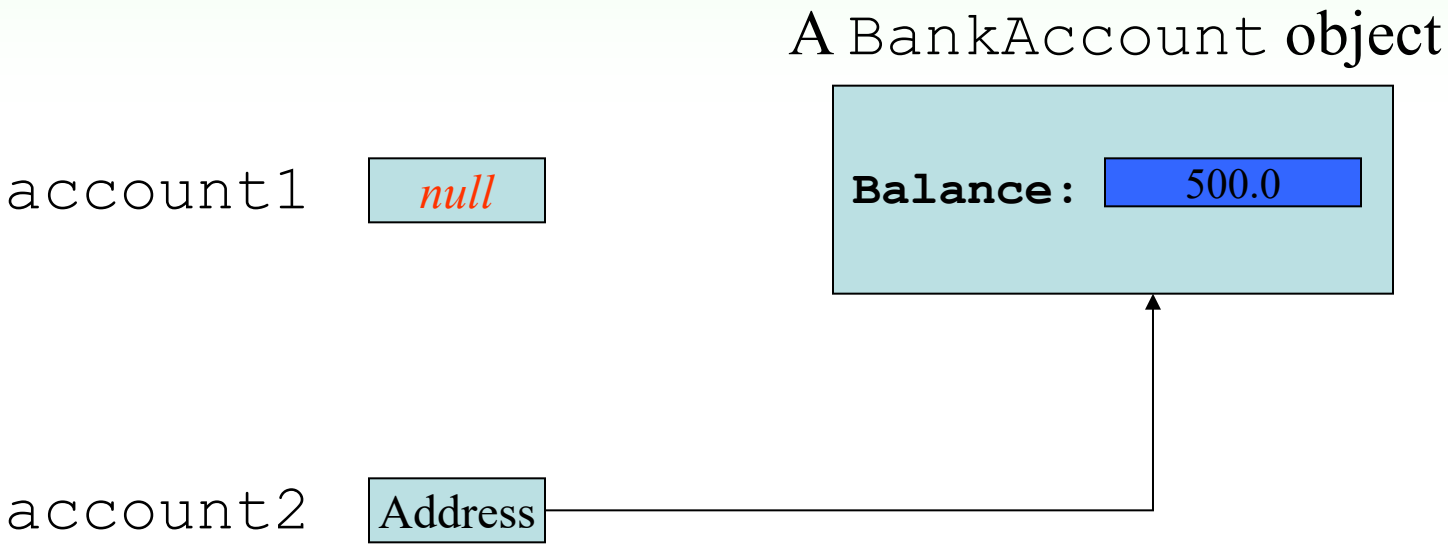
Garbage Collection



Here, both `account1` and `account2` point to the same instance of the `BankAccount` class.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Garbage Collection

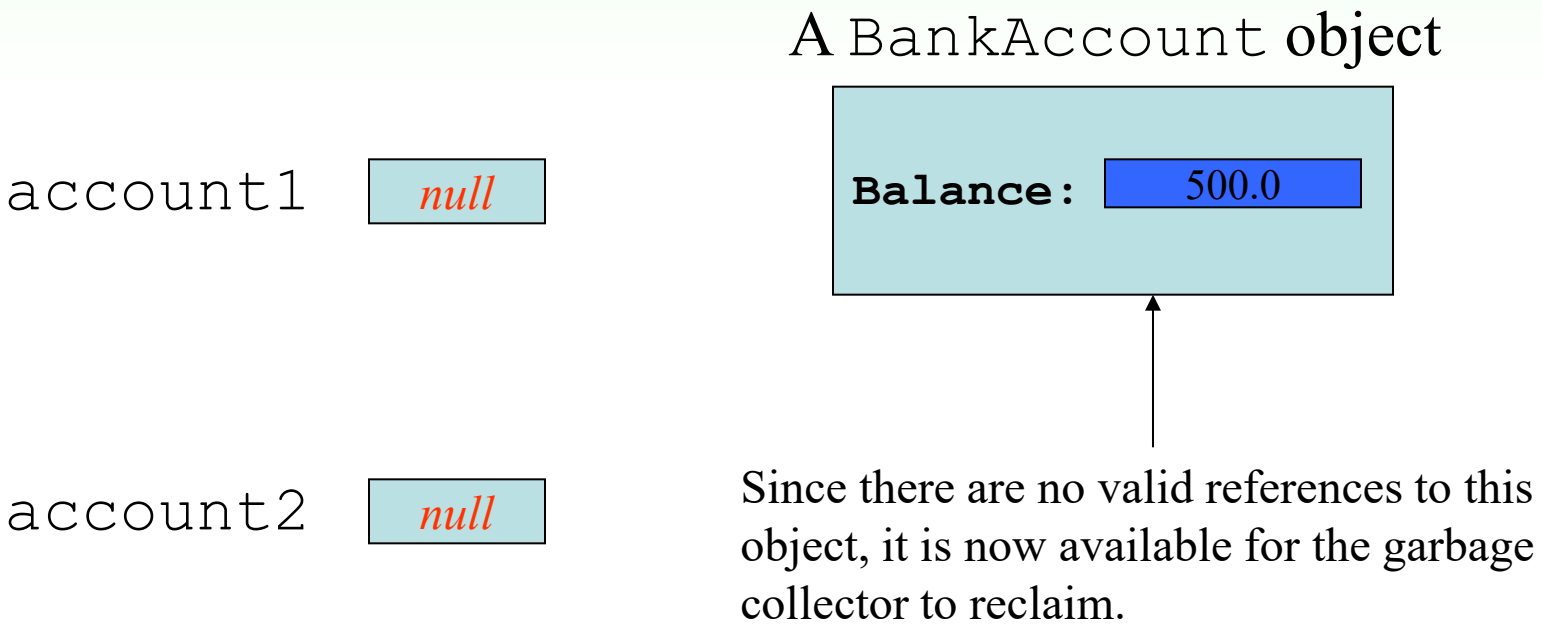


However, by running the statement: `account1 = null;`
only `account2` will be pointing to the object.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

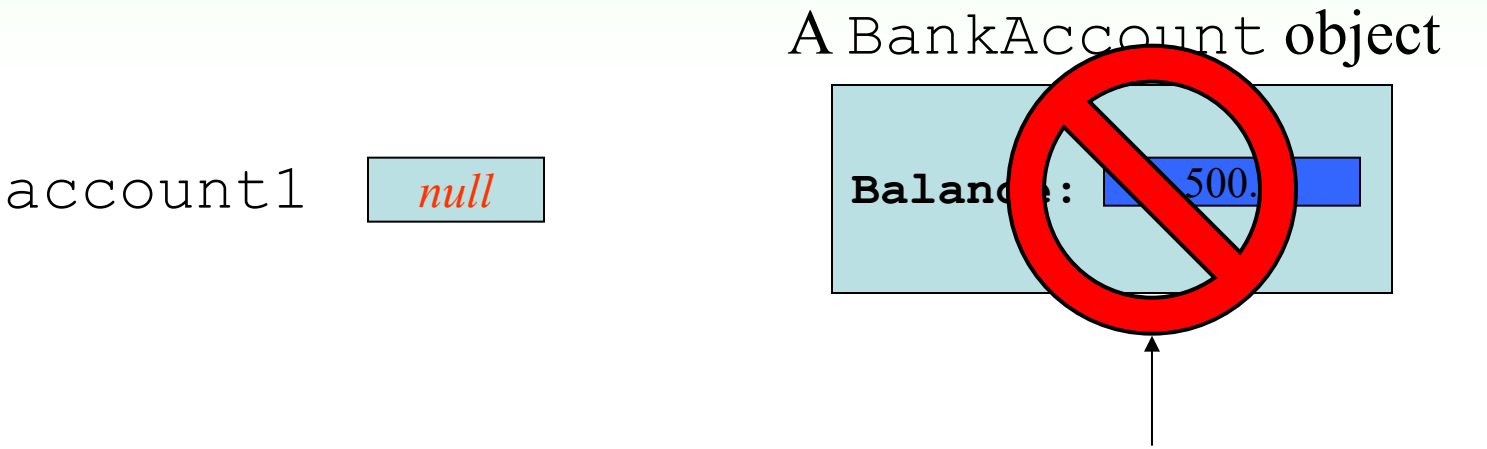
Garbage Collection



If we now run the statement: `account2 = null;`
neither `account1` or `account2` will be pointing to the object.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Garbage Collection



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `finalize` Method

- If a method with the signature:

```
public void finalize(){…}
```

  is included in a class, it will run just prior to the garbage collector reclaiming its memory.
- The garbage collector is a background thread that runs periodically.
- It cannot be determined when the `finalize` method will actually be run.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Class Collaboration

- Collaboration – two classes interact with each other
- If an object is to collaborate with another object, it must know something about the second object's methods and how to call them
- If we design a class `StockPurchase` that collaborates with the `Stock` class (previously defined), we define it to create and manipulate a `Stock` object

See examples: StockPurchase.java, StockTrader.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

CRC Cards

- Class, Responsibilities and Collaborations (CRC) cards are useful for determining and documenting a class's responsibilities
  - The things a class is responsible for knowing
  - The actions a class is responsible for doing
- CRC Card Layout (Example for class Stock)

Stock	
Know stock to purchase	Stock class
Know number of shares	None
Calculate cost of purchase	Stock class
Etc.	None or class name

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.