Chapter 6 Review

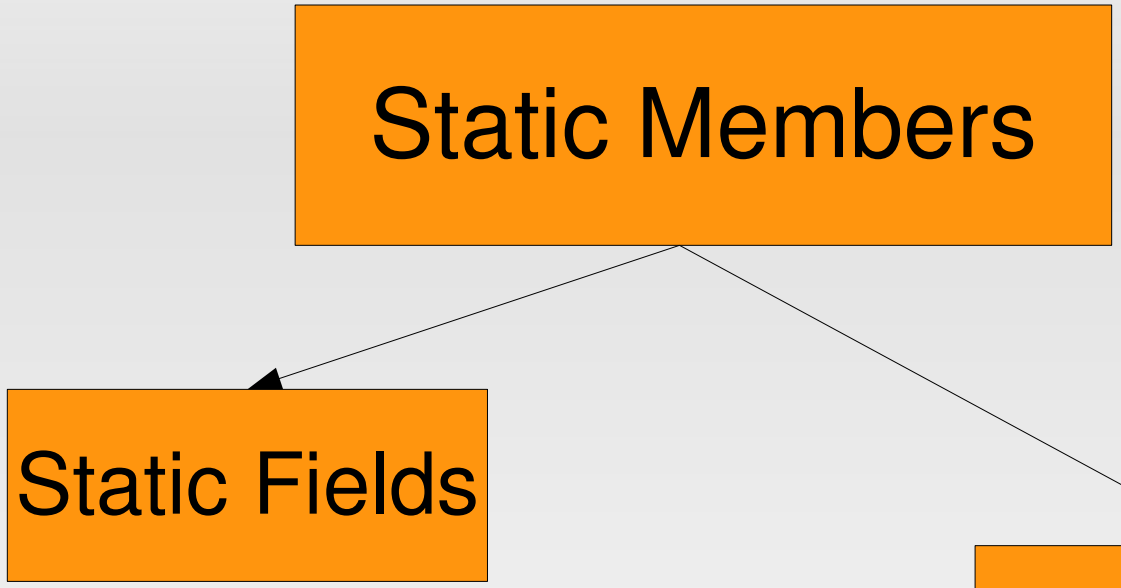
Freshta Popalyar

Date:

Contents

- Static class members
- Overloaded methods
- Overloaded constructors
- Passing objects as arguments to methods
- Returning objects from methods
- The toString method
- Writing an equals method
- Methods that copy objects
- Aggregation
- The this reference variable
- Inner classes
- Enumerated types
- Garbage collection

Static Class Members




```
Private static int instanceCount=0;
```

```
public static double mileToKilo(double miles)
   {
      return miles * 1.609;
   }
```

Overloaded Methods

- Two or more methods in a class may have the same name as long as their signatures are different.

```
Public static int square(int n){
   return n * n;
}
public static double square(double n){
   return n * n;
}
```

Method Signatures

- square (int)
- square (double)

Overloaded Constructors

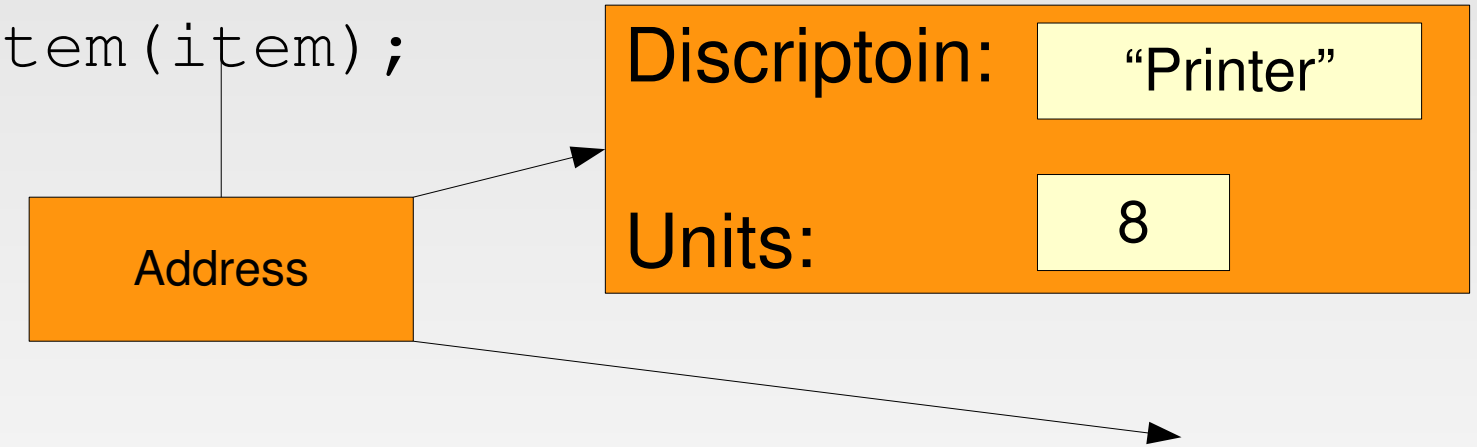
- A class can have more than one constructor:

```
public class Rectangle
   {
      private double length, width;
      public Rectangle(){
         length = 0.0;
         width = 0.0;
      }
      public Rectangle(double len, double w){
         length = len;
         width = w;
      }
```

Passing Objects as Method Args

- To pass an object as a method argument you pass an object reference

```
displayItem(item);
```




```
Public static void displayItem(InventoryItem i)
{
}
```

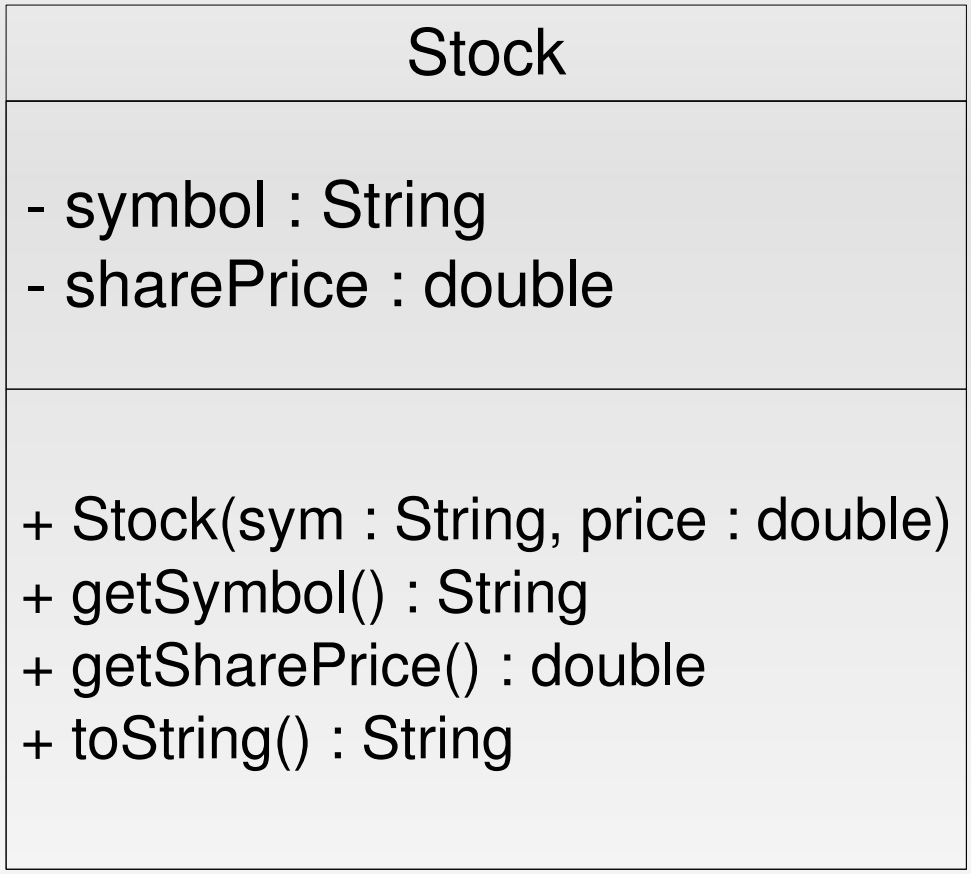
Returning Objects from Methods

- A method can return a reference to an object

```
item = getData();

public static InventoryItem getData()
{
..........
return new InventoryItem(desc,units);
}
```

Writing a toString() Method

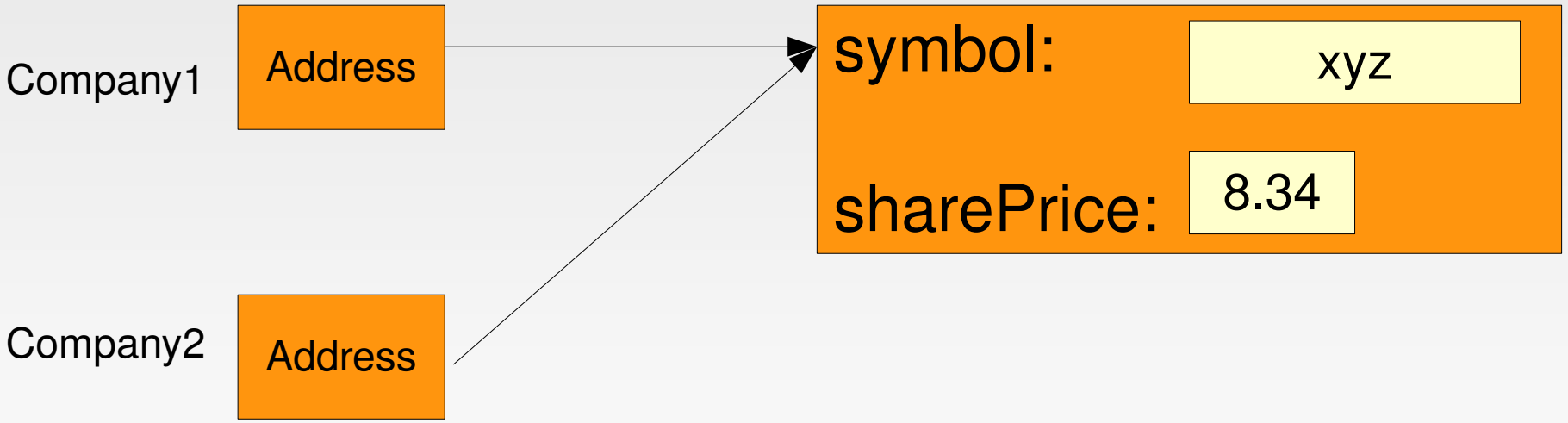


Writing an equals() Method

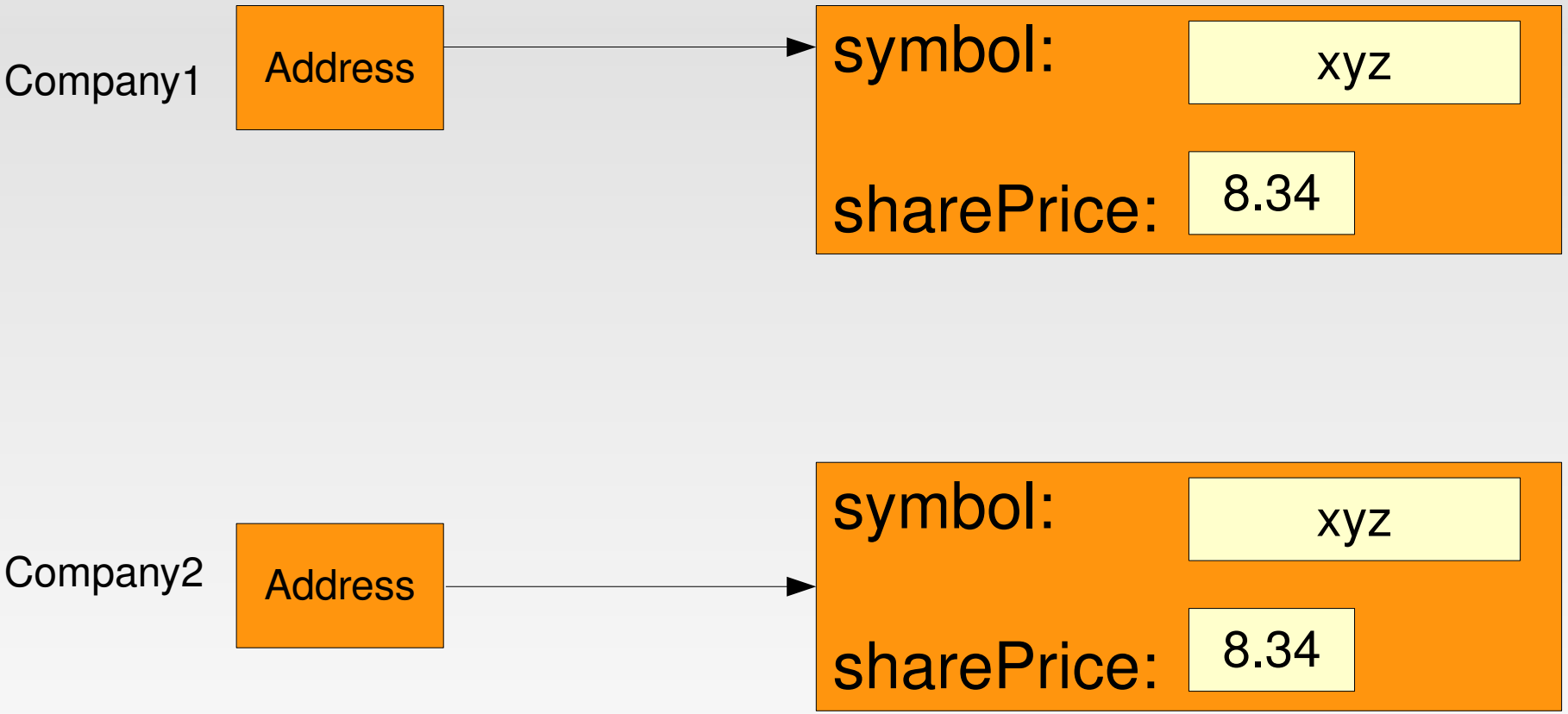
- You can not determine whether two objects contain the same data by comparing them with == operator. Instead, the class must have a method such as equals() for comparing the contents of objects.

Methods that Copy Objects

- You can prevent duplication of objects by equipping the class with a method that returns a copy of an existing object.



Cont...


Company1
Address
symbol:
xyz
sharePrice:
8.34
Company2
Address
symbol:
xyz
sharePrice:
8.34

The Copy Method and Copy Constructor

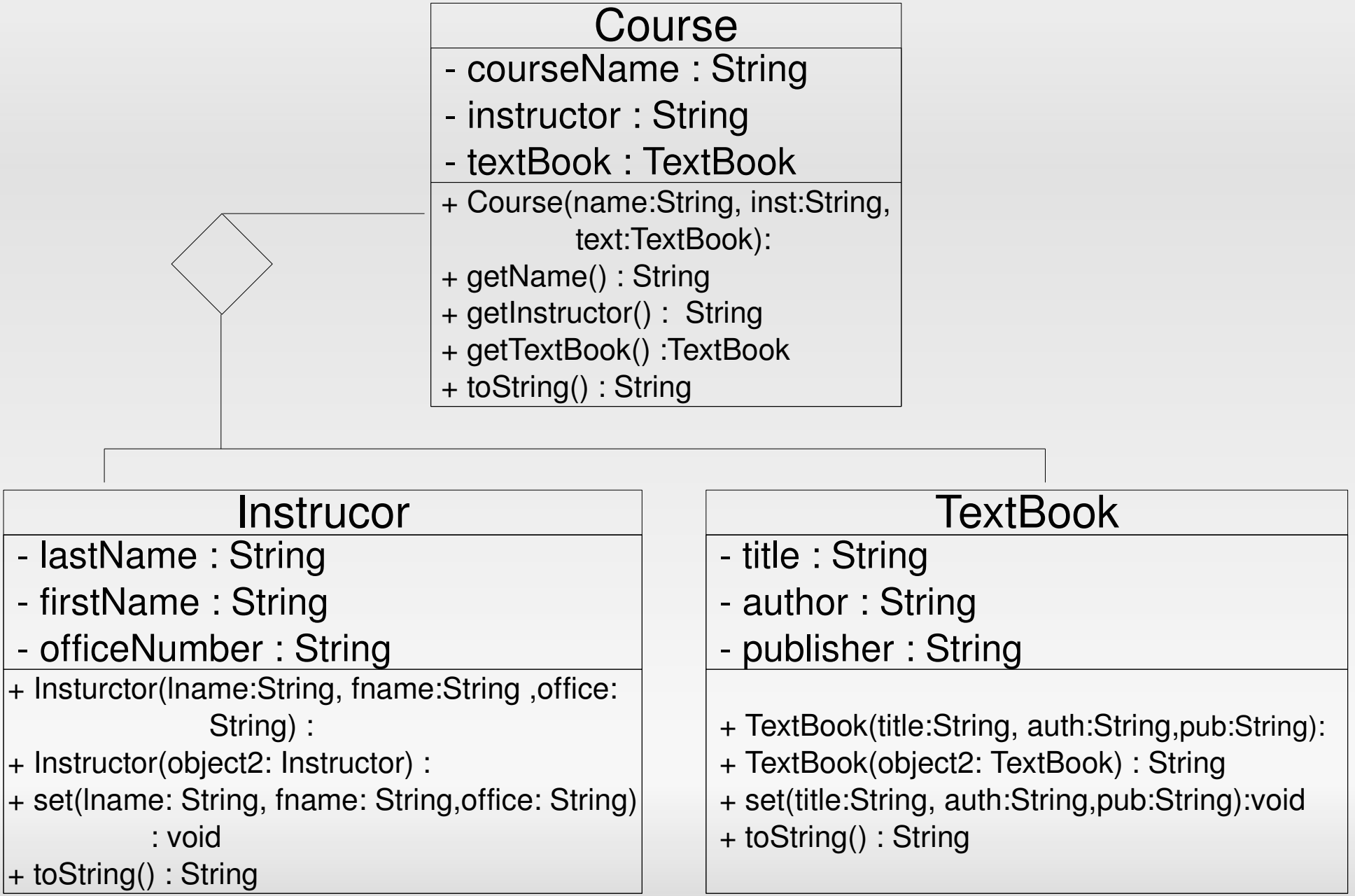
The Copy Method:

```
public Stock copy(){
   Stock copyObject=newStock(symbol,sharePrice);
   return copyObject;
}
```

The Copy Constructor:

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

Aggregation



Aggregation

- The “has a” relationship
  - The course *has an* Instructor
  - The course *has a* text book
- Security issues with Aggregate classes
  - Perform deep copies when creating field objects
  - Return copies of field objects, not the original objects

Performing Deep Copies

- instructor = new Instructor(instr);
- textBook = new TextBook(text);

- public Course(String name, Instructor instr, TextBook text){

```
    //the following assigned values
    // can create security holes
    courseName = name;
    instructor = instr;
    textBook = text;
}
```

Returning Copies of Objects

- public Instructor getInstructor(){

  retrun new Instructor(instructor);

  }

- public Instructor getInstructor(){

  retrun instructor;

  }

Null References

- Avoid using null references, it can cause the program to crash.
- ```
  FullName name=new FullName();

  System.out.println(name.length());
  ```
- There are two ways of avoiding such situations
  - Write a constructor that initializes the fields to empty strings.
  - Make sure the fields are not null with code, like using if statements.
  ```

The this Reference Variable

- `symbol.equals(object2.symbol)`
- `this.symbol.equals(object2.symbol)`

continued...

- Using “this” to overcome shadowing
- ```
  public stock(String sym, double Price){
  ```

  ```
  symbol = sym;
  ```

  ```
  sharePrice = price;    }
  ```
- We can use the following instead
- ```
  public stock(String symbol, double sharePrice){
  ```

  ```
  this.symbol = symbol;
  ```

  ```
  this.sharePrice = sharePrice;
  ```

  ```
  }
  ```

continued...

- Using “this” to call overloaded constructors
- ```
  public stock(String sym, double Price){
    symbol = sym;
    sharePrice = price;    }
  ```
- Assume we have another constructor
- ```
  public stock(String sym){
    this(sym, 0.0);   }
  ```
- *this* can be used to call the constructor from another constructor in the same class
- It must be the first statement of constructor

Inner Classes

- Inner class is a class that is defined inside another class definition
- An outer class can access the public members of inner class.
- A private inner class is not visible or accessible to code outside the outer class.
- An inner class can access the private members of the outer class.

Enumerated Types

- An Enumerated Data Type consists of a set of predefined values. You can use the data type to create variables that can hold only the values that belong to the enumerated data type.
- `enum Day {SATURDAY, SUNDAY, MONDAY TUESDAY, WEDNESDAY, THURSDAY, FRIDAY}`
- `Day workDay = Day.WEDNESDAY;`
- `System.out.println(workday.ordinal());`

Garbage Collection