ICT104

Program Design and Development

Lecture 3– A second look at Classes and Objects (cont'd)

Adopted from: Gaddis & Gaddis (2019) Starting Out with Java: From Control Structures through Objects, 7th Edition.

King's Own Institute

Focus for this week

A Second Look at Classes and Objects (cont'd)

- Returning Objects from Methods
- The `toString()` Method
- The `equals()` method
- Methods that copy objects
- Copy constructors
- Aggregation
- Returning references to private fields
- NULL references
- `this` reference
- Enumerated types

Activity 1:
Revision Exercise

List any three concepts which you can remember from last week’s class

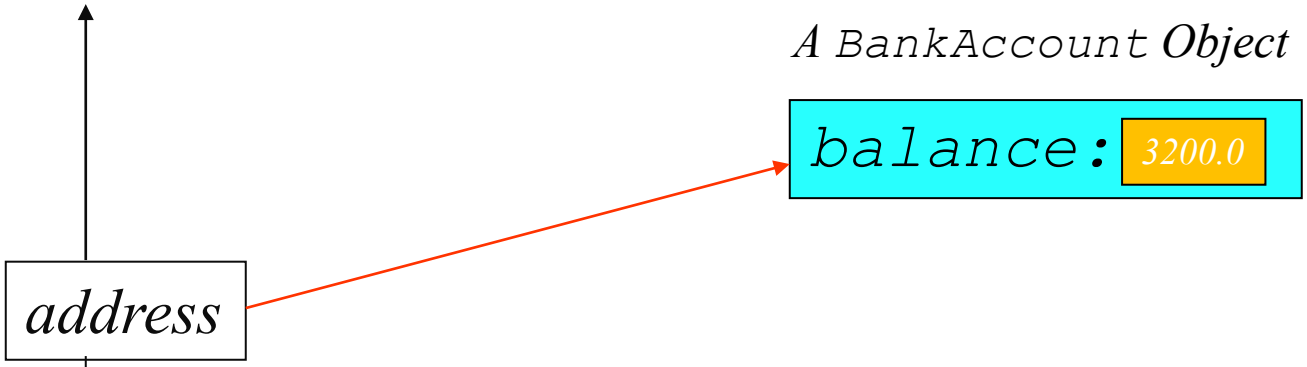
Returning Objects From Methods

- 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**
- Method return type:

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

Returning Objects from Methods

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




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

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);
```

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);
```

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

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)

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

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

Activity 2: Poll

1. Statement: To compare two objects in a class we need to write an equals method that will make a field by field compare of the two objects.

- ◯ TRUE
- ◯ FALSE

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

Copy Constructors

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

```
public Stock(Stock object2)
{
      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);
```

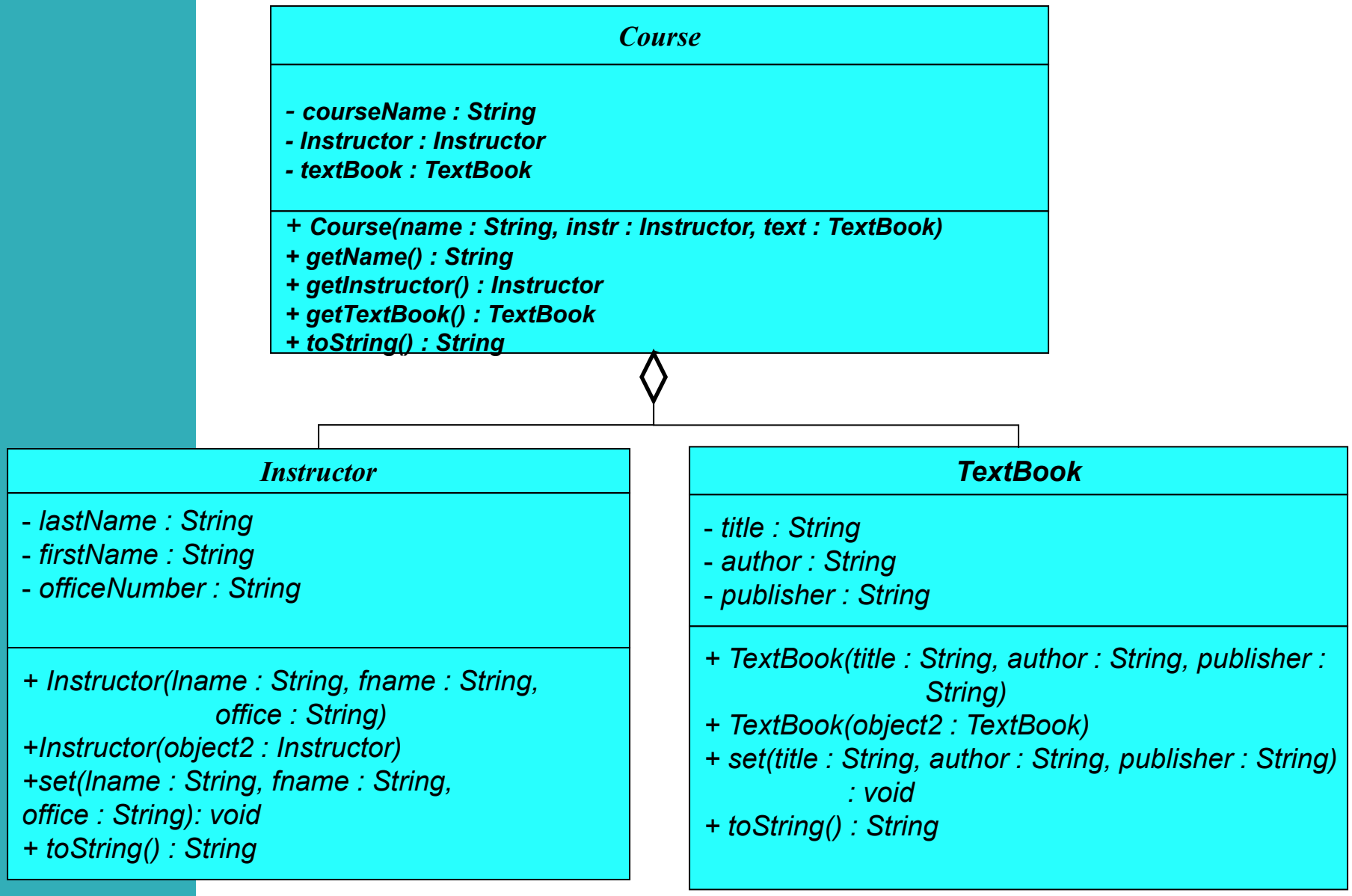
Activity 3: Discussion question

What is a constructor? Can it be overloaded? Give an example to explain your answer

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

Aggregation in UML Diagrams



Activity 4: Poll

1. The general layout of a UML diagram is a box that is divided into three sections. The top section has the _____; the middle section holds _____; the bottom section holds _____.

- ◯ class name; attributes or fields; methods
- ◯ class name; object name; methods
- ◯ object name; attributes or fields; methods
- ◯ object name; methods; attributes or fields

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

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())
}
```

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

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

Activity 5: Poll

1. When the "this" variable is used to call a constructor:

- ◯ it must be the last statement in the constructor making the call
- ◯ it can be anywhere in the constructor making the call
- ◯ you cannot use the this variable in a constructor call
- ◯ it must be the first statement in the constructor making the call

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;
```

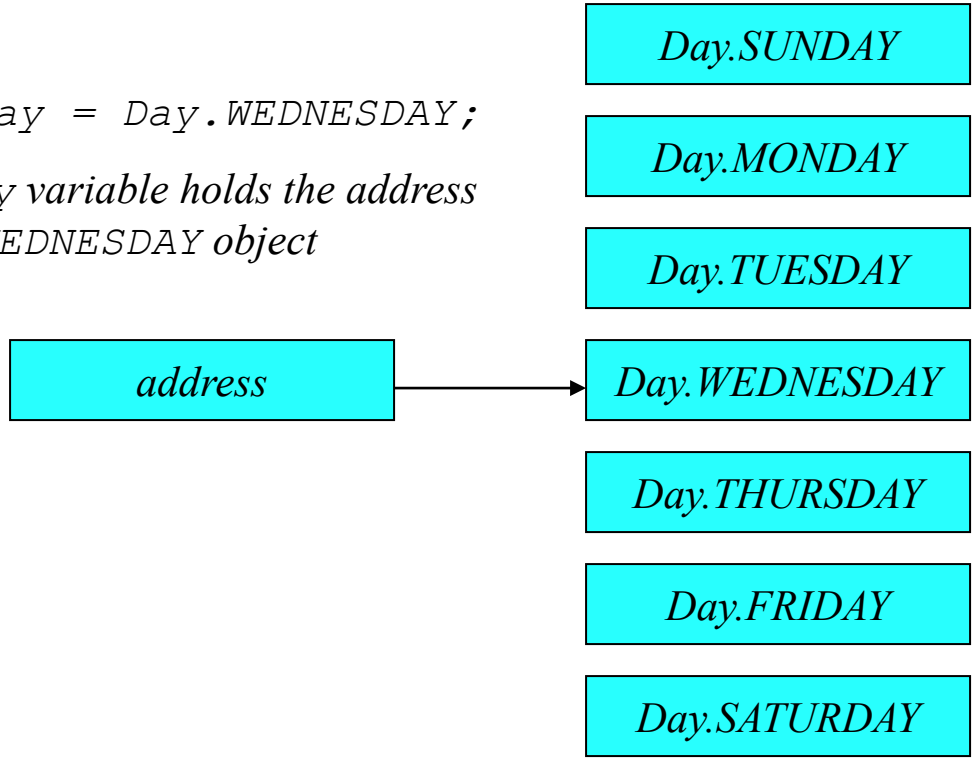
Enumerated Types

- 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*



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

Enumerated Types - Switching

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

Activity 6: Poll

1. Look at the following declaration:

enum Tree { OAK, MAPLE, PINE }
What is the ordinal value of the MAPLE enum constant?

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ Tree.MAPLE

Activity 7: Poll

1. Statement I: If you write a toString method for a class, Java will automatically call the method any time you concatenate an object of the class with a string.
Statement II: Enum constants have a toString method

- ◯ Both Statement I and Statement II are TRUE.
- ◯ Both Statement I and Statement II are FALSE.
- ◯ Statement I is TRUE and Statement II is FALSE.
- ◯ Statement I is FALSE and Statement II is TRUE.

Summary of todays lesson

A Second Look at Classes and Objects (cont'd)

- Returning Objects from Methods
- The `toString()` Method
- The `equals()` method
- Methods that copy objects
- Copy constructors
- Aggregation
- Returning references to private fields
- NULL references
- `this` reference
- Enumerated types

Activity 8:
Reflection Exercise

List any four concepts you have learnt in today's lesson

Activity 9: Homework Exercise

Write a program using either Eclipse or NetBeans to implement any one concept you have learnt in today's lesson