UNIVERSIDAD NACIONAL DE INGENIERIA | FACULTAD DE CIENCIAS

Unit 5:

Advanced OO Concepts

Juan Espejo

Go to Classroom

September 10, 2018

Content

- Constructors
- Error Handling
- The Importance of Scope
- Operator Overloading
- Multiple Inheritance
- Object Operations

Constructors

```
public Cabbie(){
    /* code to construct the object */
}
```

Figure: An example.[1]

```
public int Cabbie(){
    /* code to construct the object */
}
```

Figure: A counterexample.[2]

[1] Page 53 of [1]
[2] Page 54 of [1]

Constructors

```
Cabbie myCabbie = new Cabbie();
```

Figure: When is a constructor called?[3]

[3] Page 54 of [1]

Constructors

```
public class Cabbie {
    int id;

    public Cabbie(){
        id = 1;
    }
}
```

Figure: What is inside a constructor?

Constructors

```
public Cabbie(){
    super();
}
```

Figure: The **default constructor.**[4]

Providing a constructor: the general rule is that you should always provide a constructor, even if you do not plan to do anything inside it.[5]

[4] Page 55 [1]
[5] Page 55 of [1]

Constructors

```
public class Cabbie {
    int id;

    public Cabbie(){
        id = 1;
    }

    public Cabbie(int number){
        id = number
    }
}
```

Figure: Multiple constructors of a Cabbie object.

Constructors

```
// different parameter list
public void getCab (String cabbieName);

// different parameter list
public void getCab (int numberOfPassengers);
```

Figure: Overloading methods.[6]

[6] Page 56 of [1]

Constructors

Signature

public String getRecord(int key)

Signature = getRecord (int key)
method name + parameter list

Figure: The components of a signature.[7]

[7] Figure 3.1 of [1]

Constructors

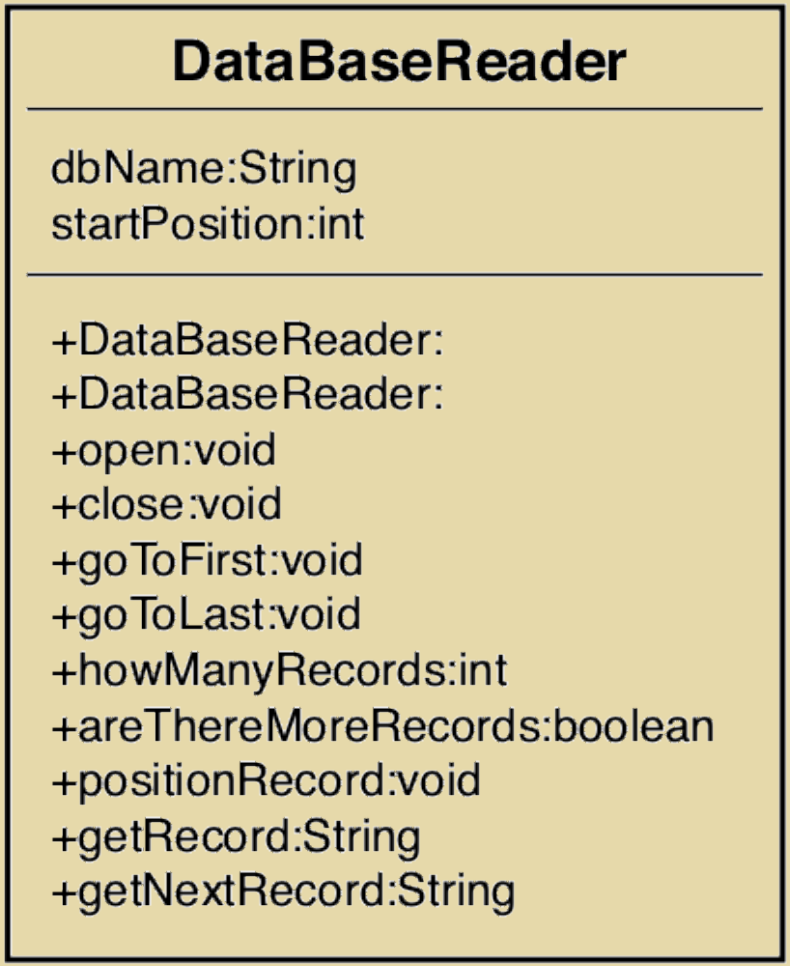

Figure: The `DataBaseReader` diagram.[8]

[8] Figure 3.2 of [1]

Constructors

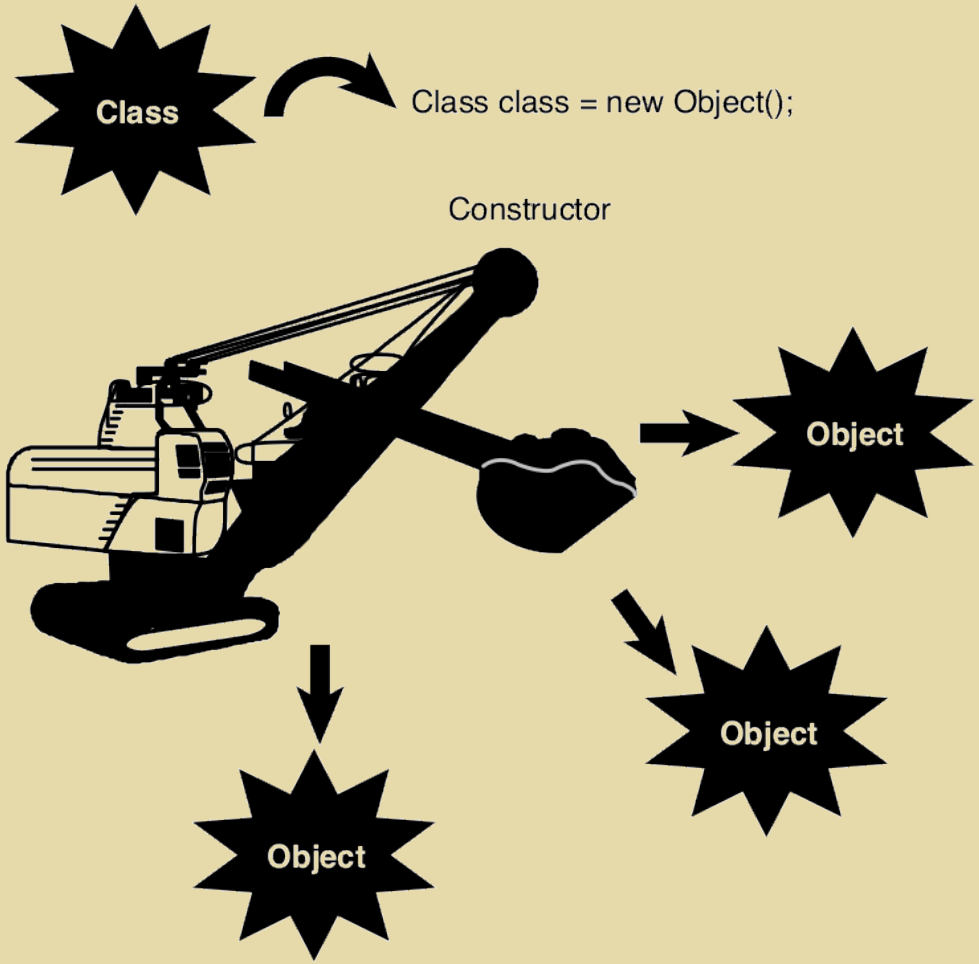

Figure: Creating a new object.[9]

[9] Figure 3.3 of [1]

Error Handling

- Ignoring the Problem
- Checking for Problems and Aborting the Application
- Checking for Problems and Attempting to Recover
- Throwing an Exception

Error Handling

```
if (a == 0) a = 1;

c = b/a;
```

Figure: However, setting `a` to `1` might not be a proper solution because the result would be incorrect.[11]

[10] Page 61 of [1]
[11] Page 61 of [1]

Error Handling

Figure: Catching an exception.[12]

[12] Figure 3.5 of [1]

The Importance of Scope

Methods represent the behaviors of an object; the state of the object is represented by attributes. There are three types of attributes:

- Local attributes
- Object attributes
- Class attributes[13]

[13] Page 64 of [1]

The Importance of Scope

```
public class Number {

    public method1() {
        int count;
    }

    public method2() {
        int count;
    }

}
```

Figure: When `method1` terminates, the copy of `count` is removed.[14]

[14] Page 65 of [1]

The Importance of Scope

```
public class Number {

    int count;    // available to both method1 and method2

    public method1() {
        count = 1;
    }

    public method2() {
        count = 2;
    }

}
```

Figure: Note here that the class attribute `count` is declared outside the scope of both `method1` and `method2`.[16]

[15] Page 66 of [1]
[16] Page 66 of [1]

The Importance of Scope

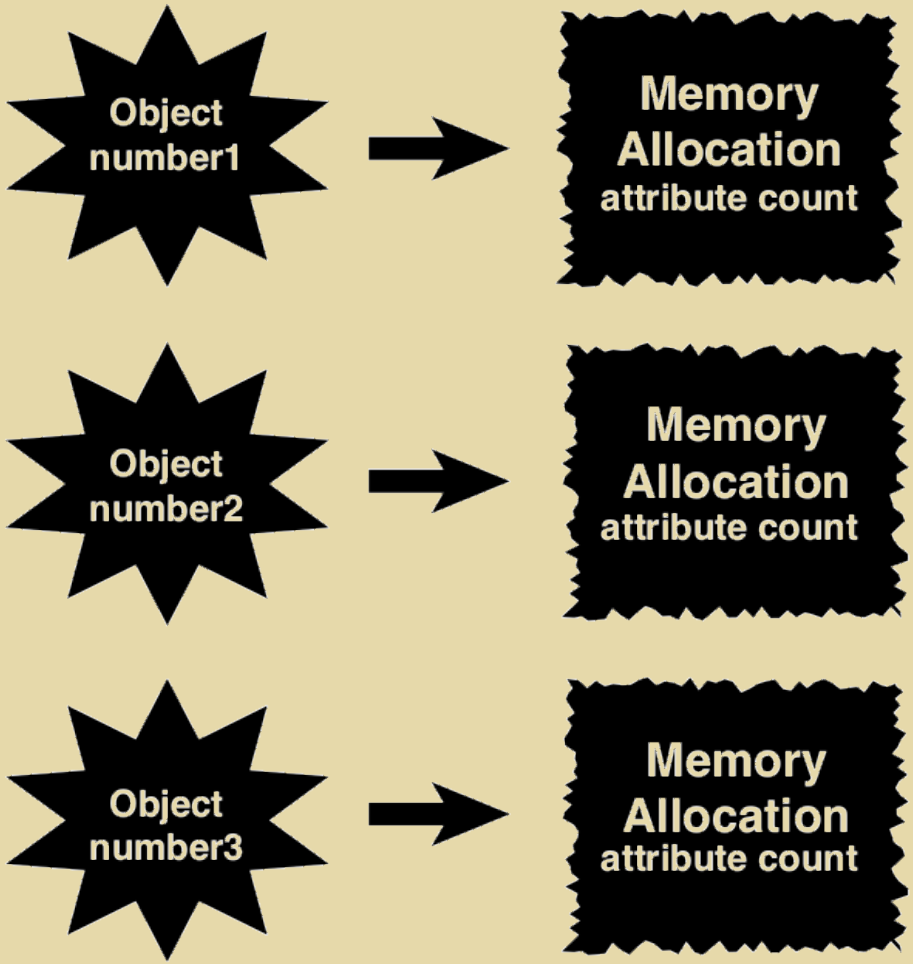

Figure: Object attributes.[17]

[17] Figure 3.6 of [1]

The Importance of Scope

```
public method1() {
        int count;

        this.count = 1;
}
```

Figure: The keyword `this` is a reference to the current object.[18]

[18] Page 67 of [1]

The Importance of Scope

```
public class Number {

    static int count;

    public method1() {
    }

}
```

Figure: By declaring `count` as static, this attribute is allocated a single piece of memory for all objects instantiated from the class.[20]

[19] Page 68 of [1]
[20] Page 68 of [1]

The Importance of Scope

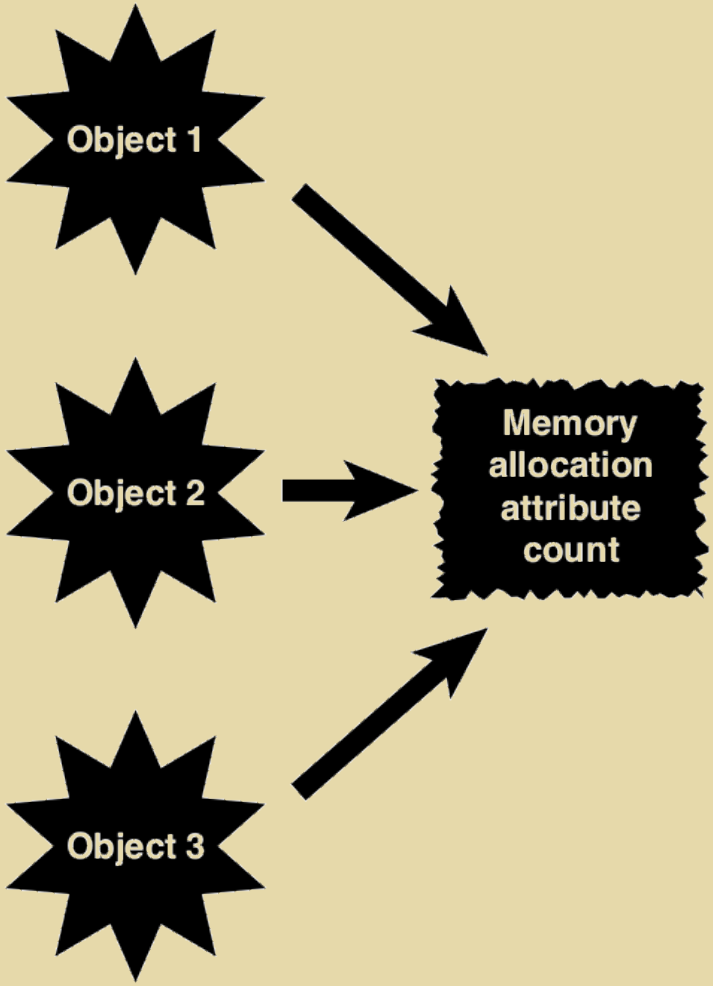

Figure: Class attributes.[21]

[21] Figure 3.7 of [1]

Operator Overloading

```
String firstName = "Joe", lastName = "Smith";

String Name = firstName + " " + lastName;
```

Figure: String concatenation occurs when two separate strings are combined to create a new, single string.[23]

[22] Page 69 of [1]
[23] Page 69 of [1]

Multiple Inheritance

Figure: Family tree

Object Operations

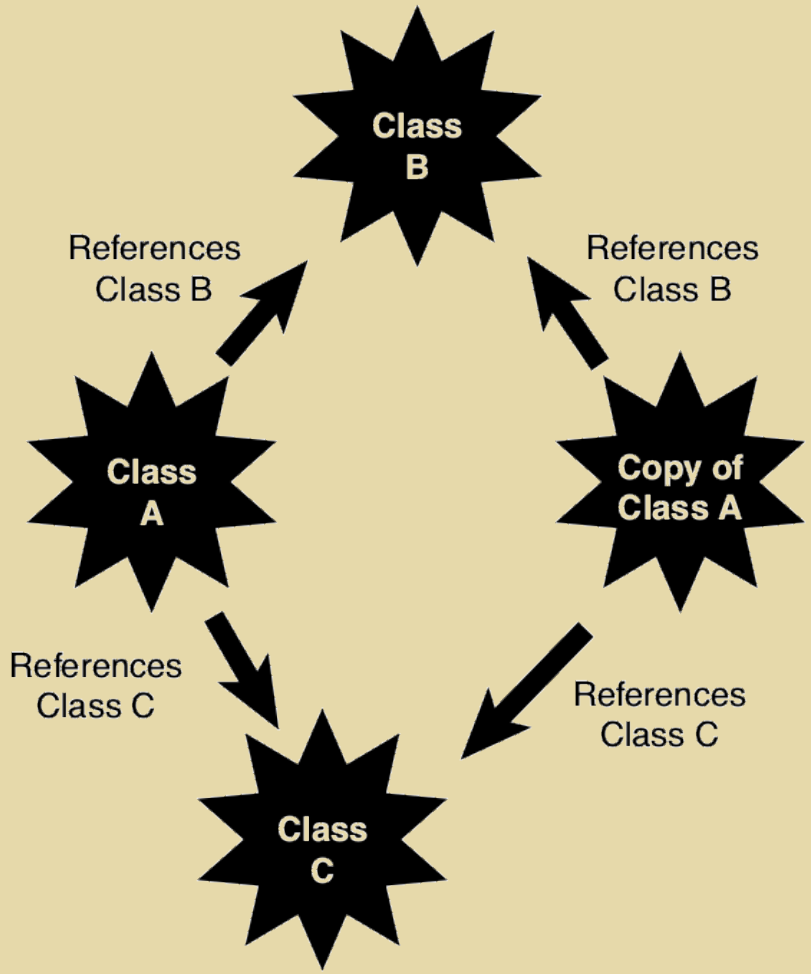

Figure: Following Object References.[24]

[24] Figure 3.8 of [1]

References

WEISFELD, M.
The Object-Oriented Thought Process, 4th ed.
Developer's Library. Addison-Wesley Professional, 2013.