Jin L.C. Guo

Content Review

by Deeksha Arya

Image Source: https://upload.wikimedia.org/wikipedia/commons/2/2b/Cepaea_nemoralis_active_pair_on_tree_trunk.jpg

What we'll (try to) cover today:

- Unit Testing
- Overriding Equals and Hashcode
- UML Diagrams
- Null Object Pattern
- Clarifying Iterable vs Iterator
- Clarifying Comparable vs Comparator

Unit Testing

Still a regular class:

(line 9) This is new ->

No main method?

```
1  package testpackage;
2
3  import static org.junit.Assert.*;
6
7  public class TestCase {
8
9      @Test
10     public void test() {
11         assertTrue(true);
12     }
13
14 }
```

Flash Light Example

Let's model a (simple) Flash Light 🔦

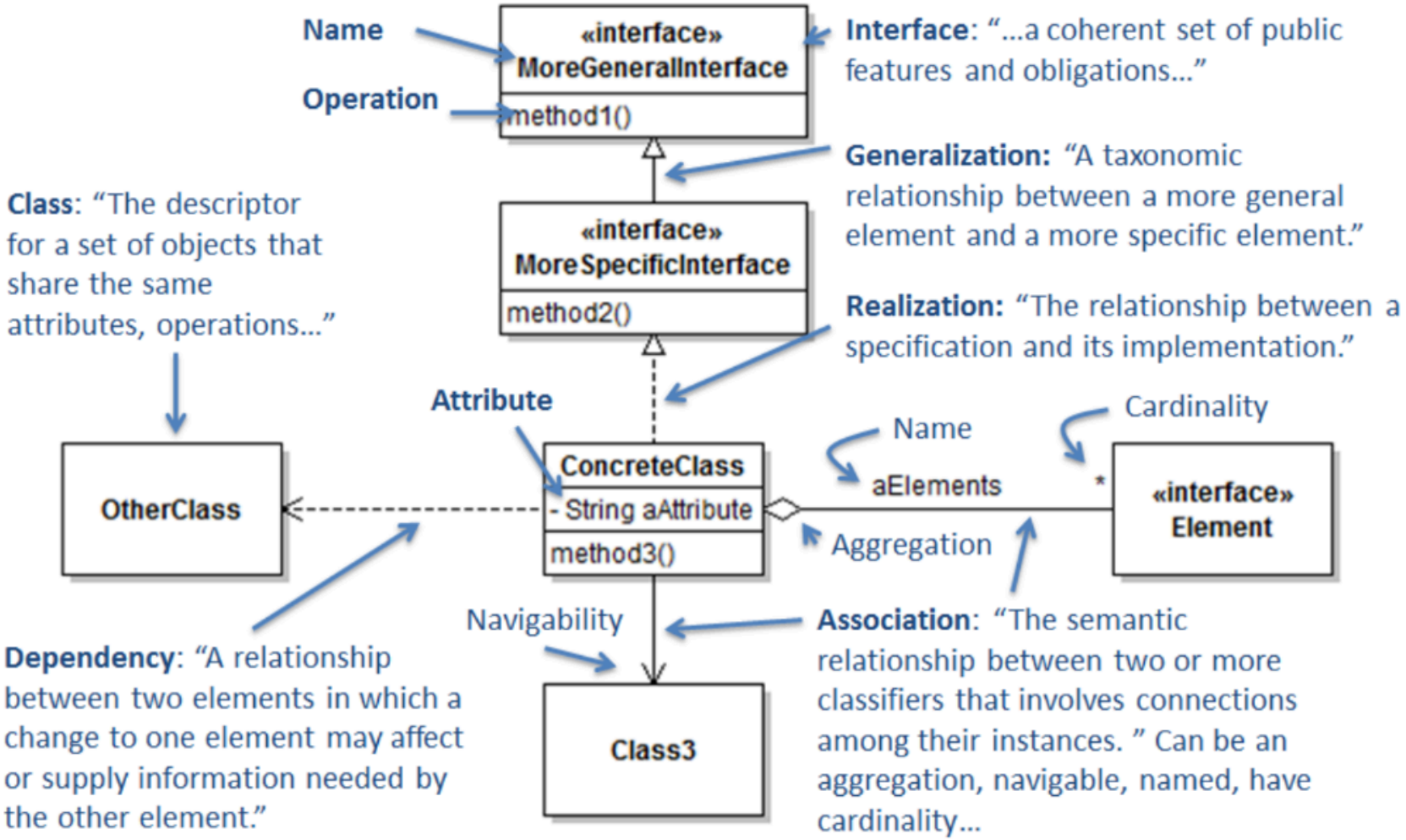
It has one button that toggles it on and off:

```
1  package custompackage;
2
3  public class FlashLight {
4
5      private boolean isOn = false;
6
7      public void pressButton() {
8          this.isOn = ! this.isOn;
9      }
10
11     public boolean isOn() {
12         return this.isOn;
13     }
14 }
```

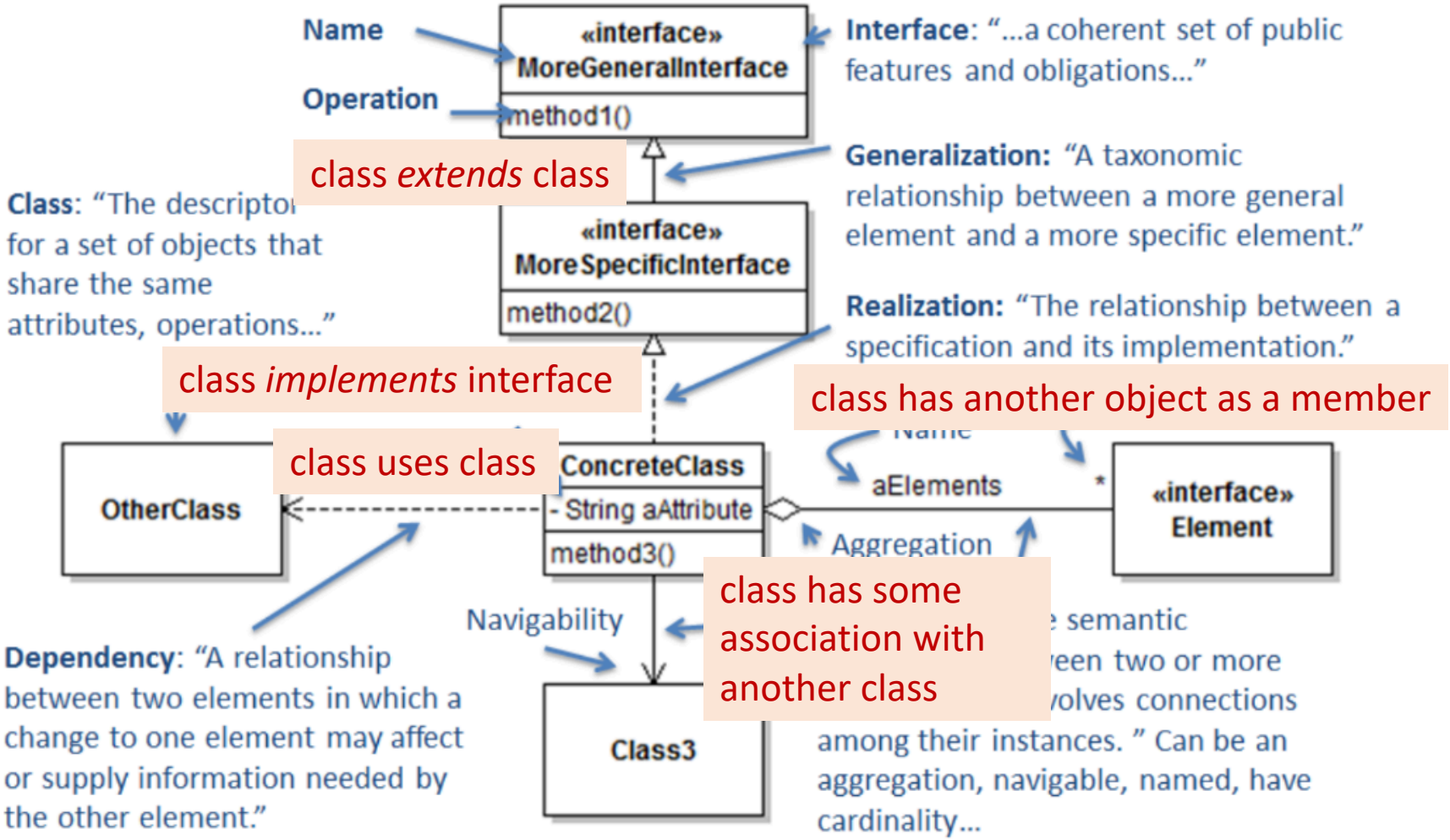
A Typical Unit Test

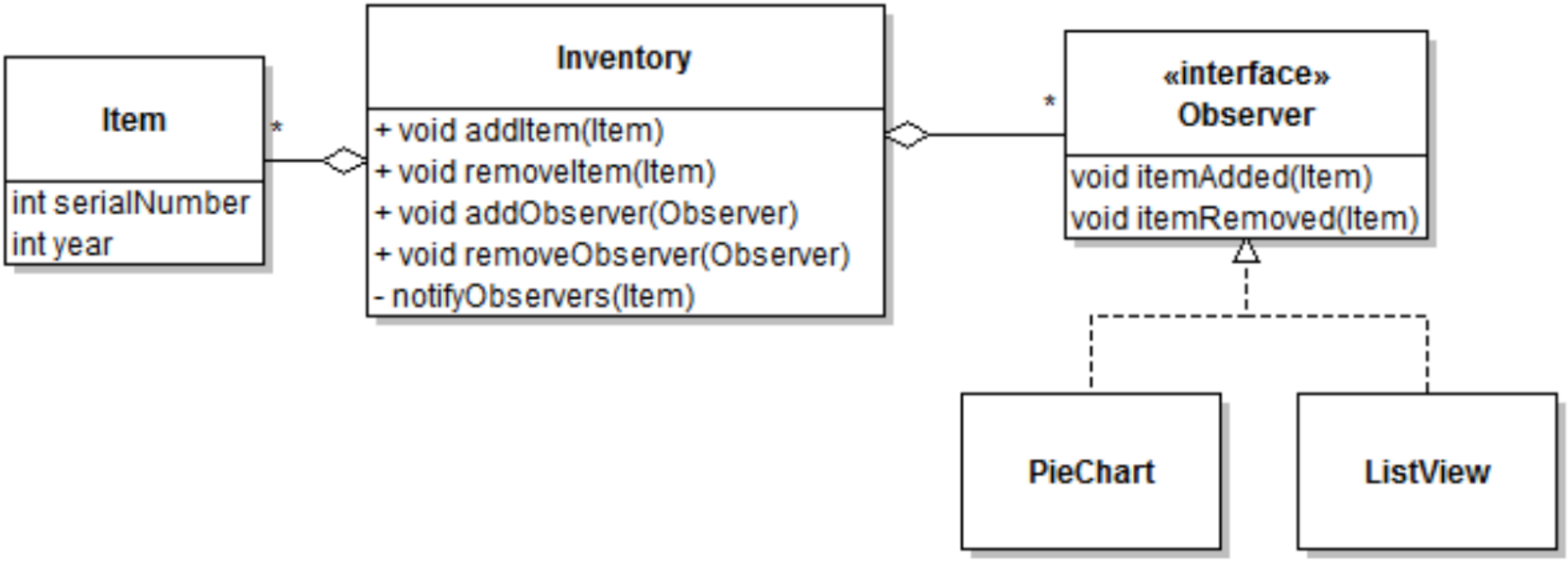
```
1  package custompackage;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5  import custompackage.FlashLight;
6
7  public class FlashLightTest {
8      @Test
9      public void testOffByDefault() {
10         FlashLight kitchenFlashLight = new FlashLight();
11         assertFalse(kitchenFlashLight.isOn());
12     }
13
14     @Test
15     public void testTurnsOnAfterOnePress() {
16         FlashLight kitchenFlashLight = new FlashLight();
17         kitchenFlashLight.pressButton();
18         assertTrue(kitchenFlashLight.isOn());
19     }
20
21     @Test
22     public void testTurnsBackOffAfterTwoPresses() {
23         FlashLight kitchenFlashLight = new FlashLight();
24         kitchenFlashLight.pressButton();
25         kitchenFlashLight.pressButton();
26         assertEquals(false, kitchenFlashLight.isOn());
27     }
28 }
29
```

Class Diagram

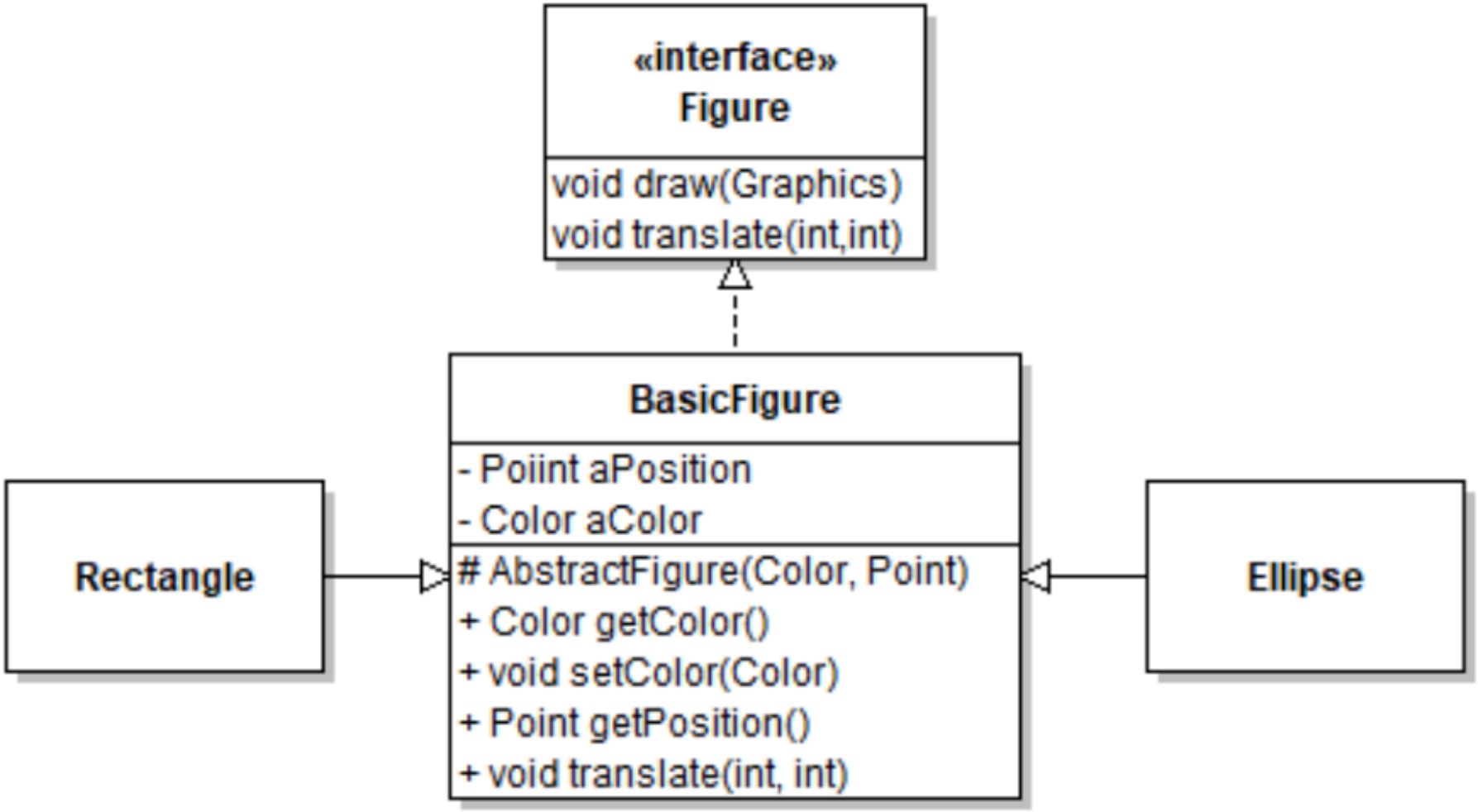


Class Diagram



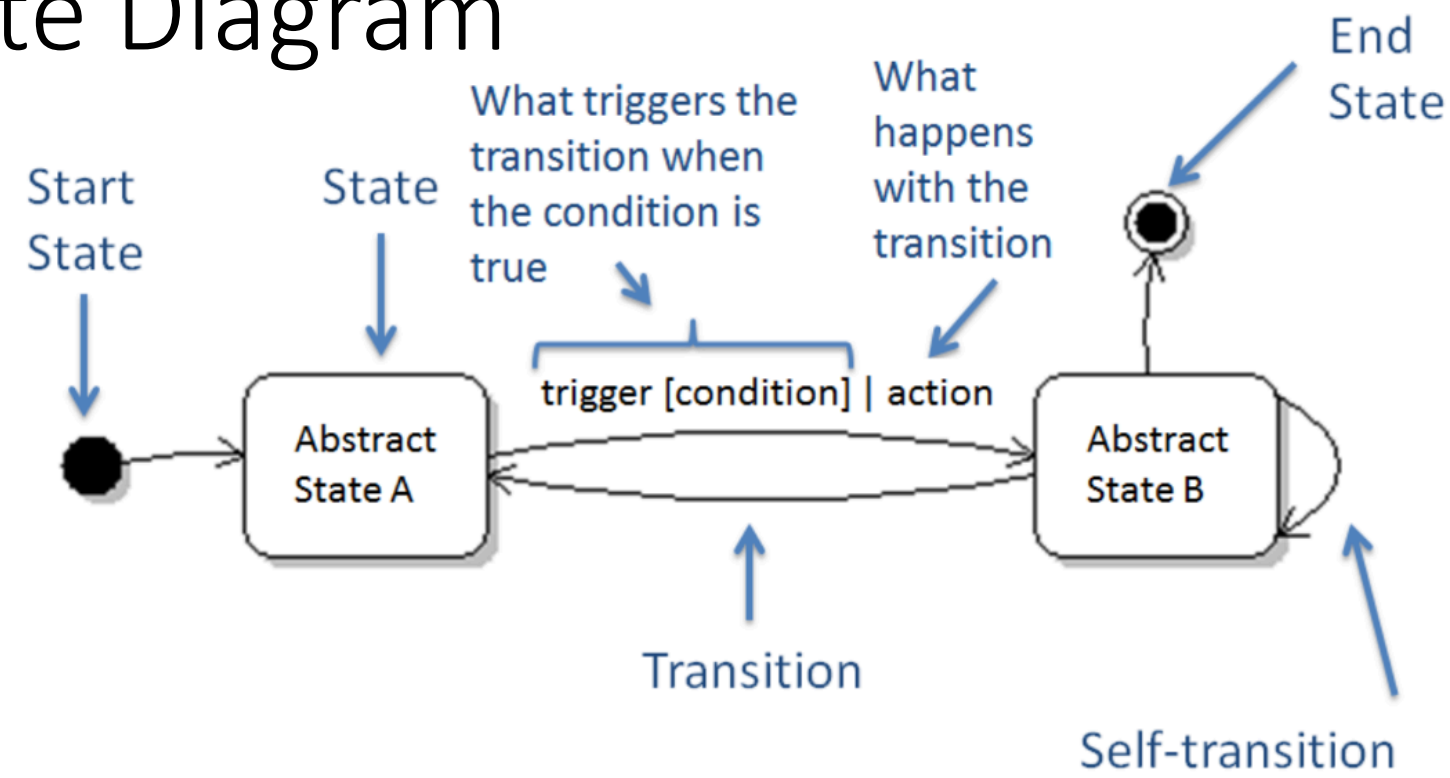


Note: Ignore what the classes are for now and concentrate on what the relationships between the classes are and how they can be translated into code.



Note: Ignore what the classes are for now and concentrate on what the relationships between the classes are and how they can be translated into code.

State Diagram



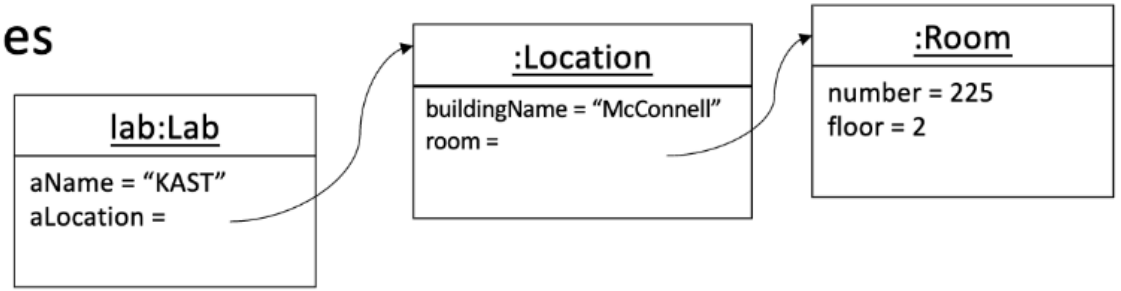
- Triggers are (almost) always **potentially state-changing** methods.
- Every abstract state is the state of an object at a point in time.
- The state is always absolute, do not make it relative. It can not be “more” or “less” than something.
- There’s not necessarily an end state, but there is always a start state (initialization). For example, a list which has add and remove methods has no “end state”.
- Exceptions are not states.
- Add ALL possible legal transitions in the diagram.

State Diagram

- Suppose you own a Warehouse.
- You can have a maximum of 100 products in your warehouse at any given point in time.
- You must have a minimum of 10 products in your warehouse at any given point in time.
- Customers can come and ask you for the price of products in your warehouse.
- Customers can also purchase items from your warehouse, as long as there are more than 10 items remaining.
- You can also restock your warehouse (i.e. add items) as long as it doesn't exceed 100.

Object Diagram

- Model the structure of the system at a specific time
- Complete or part of the system
- Include objects and data values

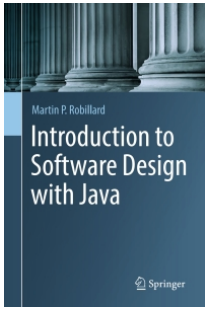


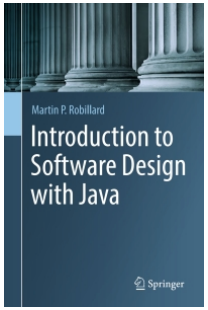
Overriding Equals and Hashcode

```
Book b1 = new Book("Introduction to Software Design with Java");
Book b2 = new Book("Introduction to Software Design with Java");
System.out.println(b1 == b2);
System.out.println(b1.equals(b2));
```

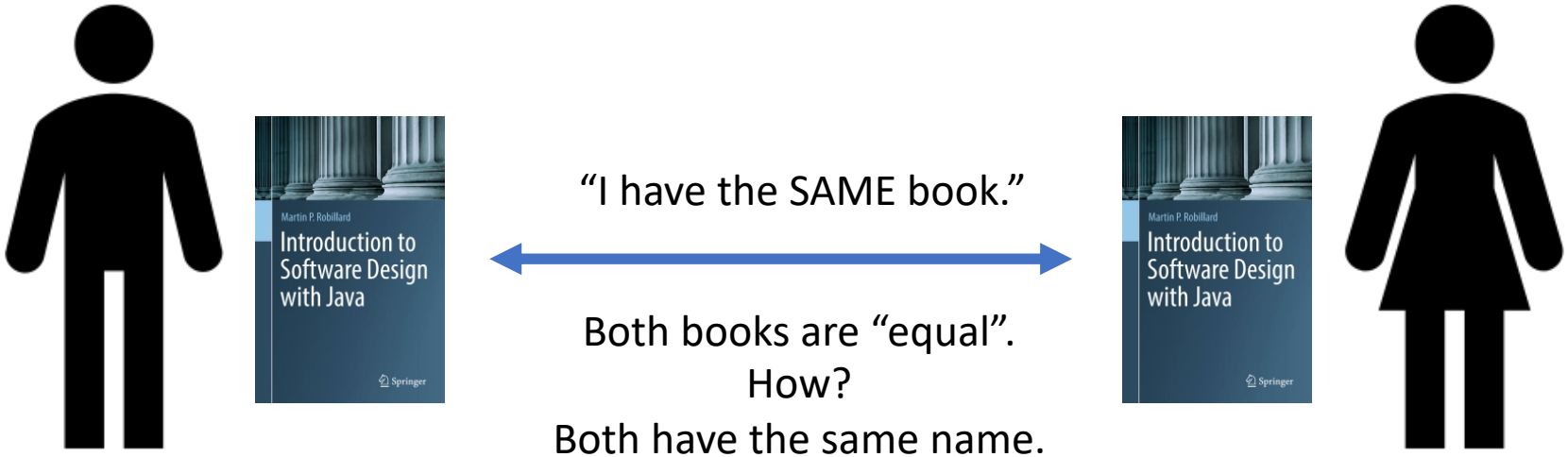
False

False





Overriding Equals and Hashcode



Both books are "equal".
How?
Both have the same name.

Overriding Equals

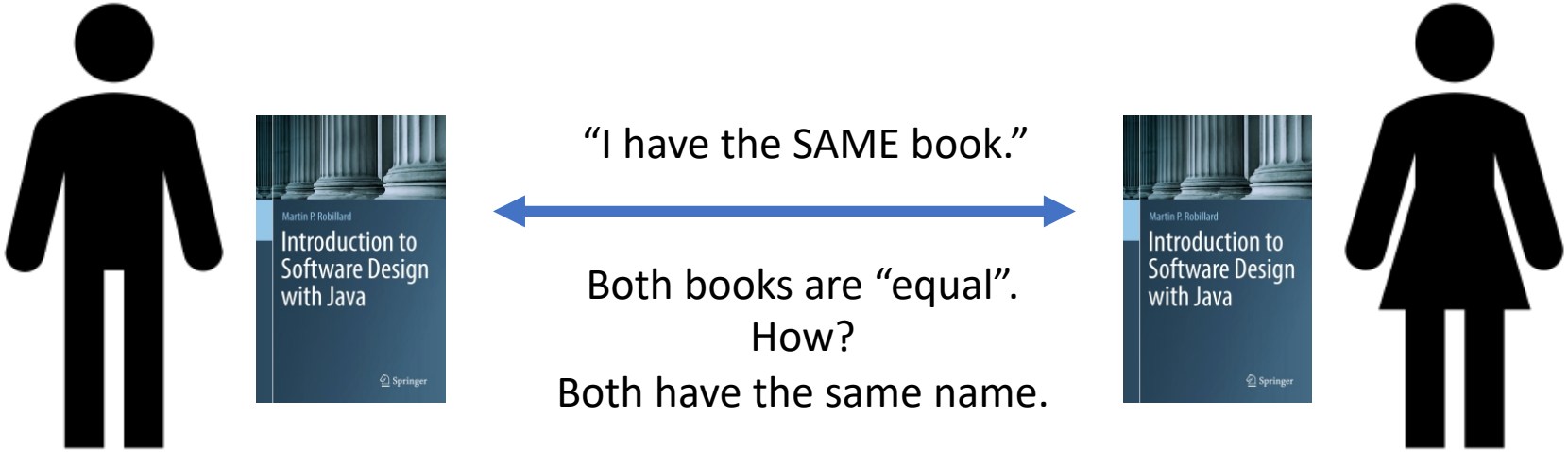
```
@Override
public boolean equals(Object obj)
{
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Book other = (Book) obj;
    if (aName == null)
    {
        if (other.aName != null)
            return false;
    }
    else if (!aName.equals(other.aName))
        return false;
    return true;
}
```

Overriding Equals and Hashcode

```
Book b1 = new Book("Introduction to Software Design with Java");
Book b2 = new Book("Introduction to Software Design with Java");
System.out.println(b1 == b2);
System.out.println(b1.equals(b2));
```

False

True



Both books are "equal".
How?
Both have the same name.

Overriding Hashcode

hashcode() is an integer value that represents the instance of a class.

Two objects which are “equal” must have the same hashcode.

But, two objects which are non-equal may still have the same hashcode.

Some logic to calculate the hashcode:

```
@Override
public int hashCode()
{
    return aName.hashCode();
}
```

```
@Override
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + ((aName == null) ? 0 : aName.hashCode());
    return result;
}
```

Null Object Pattern

- This pattern describes the “do-nothing” implementation.
- Useful when the program flow should not be interrupted if some values are undefined.

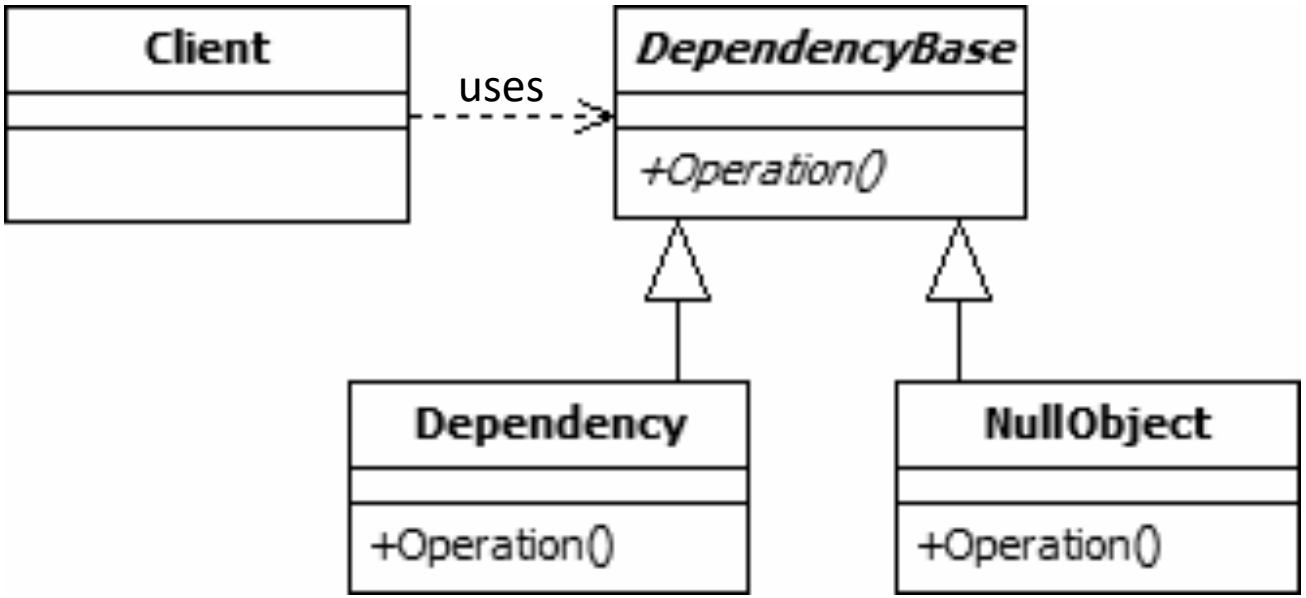
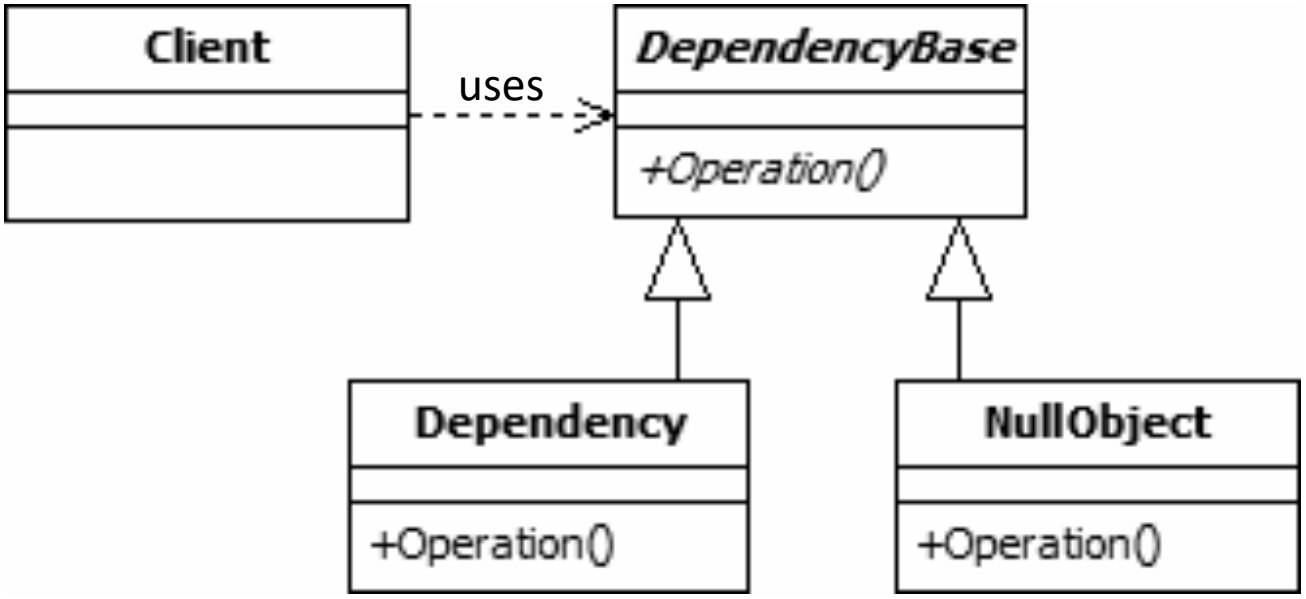


Image Source: https://www.geeksforgeeks.org/null-object-design-pattern/

Null Object Pattern

- This pattern describes the “do-nothing” implementation.
- Useful to avoid client checking for null values before execution.
- Also when the program flow should not be interrupted if some values are undefined.



Do nothing

Image Source: https://www.geeksforgeeks.org/null-object-design-pattern/

Null Object Pattern

```
public interface Router {
    void route(Message msg);
}

public class SmsRouter implements Router {
    @Override
    public void route(Message msg) {
        // implementation details
    }
}

public class JmsRouter implements Router {
    @Override
    public void route(Message msg) {
        // implementation details
    }
}

public class NullRouter implements Router {
    @Override
    public void route(Message msg) {
        // do nothing
    }
}
```

Example Source: https://www.baeldung.com/java-null-object-pattern

Null Object Pattern

What happens if the message is null?

```
public class RoutingHandler {
    public void handle(Iterable<Message> messages) {
        for (Message msg : messages) {
            Router router = RouterFactory.getRouterForMessage(msg);
            router.route(msg);
        }
    }
}
```

Method to determine which routing mechanism to use, based on the message.

- The client could have checked if `msg` is null and skipped routing it if it is null.
- But to relieve the client of this task, the `getRouterForMessage` will just return the `NullRouter`.
- So `router.route(msg)` will do nothing instead of throwing an error.

Null Object Pattern

What happens if the message is null?

```
public class RoutingHandler {
    public void handle(Iterable<Message> messages) {
        for (Message msg : messages) {
            Router router = RouterFactory.getRouterForMessage(msg);
            router.route(msg);
        }
    }
}
```

Method to determine which routing mechanism to use, based on the message.

Careful!

If `msg` should never be null,
the Null Object Pattern would hide this bug.

Iterator

Iterable

- Both are interfaces

- Stores the iterator state – is there another element to be traversed, and what it is.
- An implementing class must override the `hasNext()` and `next()` methods
- Not assigned to an object/collection of objects.

- Represents a collection that can be traversed.
- An implementing class must override the `iterator()` method, which returns an `Iterator` object using which the collection is traversed.
- A collection is said to be Iterable if it can be traversed using the Iterator.

Comparable

Comparator

- Both are interfaces

- Useful when only one type of comparison is to be made
- An implementing class must override the `compareTo()` method.

- Useful when different types of comparison is to be made and the client can choose how to compare.
- An implementing class must override the `compare()` method.
- Example of Strategy Design Pattern.

When would we use which design patterns?

Answers from the Activity:

- A Singleton Audio Manager to manage songs, playlists, etc. (Like iTunes)
- A Flyweight set of songs (or media types) so that everyone has access to the same audio files. (Useful in a shared system – say where iTunes is synced across a mobile and laptop)
- A Null Object for a new Song with no name which should “do nothing” on being played. (iTunes skips songs which don’t exist/can’t be found instead of throwing an error)
- Strategy Design Pattern to sort a list of songs based on name, artist, album name, etc. (Like iTunes has the headers of the table of songs which allow you to sort using whichever attribute.)