Chapter 6: Inheritance

EECS 1030

moodle.yorku.ca

Shape Inc.

The company Shape Inc. produces digital shapes in all shapes and sizes. Lately, Shape Inc. has received numerous complaints and lost several clients due to malfunctioning rectangles. Their rectangles are instances of the class `Rectangle` produced by the company Quadrangle and Co. Shape Inc. blames Quandrangle and Co. for their loss of revenue and considers legal action.

Shape Inc.

The company Shape Inc. produces digital shapes in all shapes and sizes. Lately, Shape Inc. has received numerous complaints and lost several clients due to malfunctioning rectangles. Their rectangles are instances of the class `Rectangle` produced by the company Quadrangle and Co. Shape Inc. blames Quandrangle and Co. for their loss of revenue and considers legal action.

Question

How do we determine who is to blame?

A complaint to Shape Inc.

A client of Shape Inc. ran the following program.

```
import com.quadrangle.Rectangle;
import org.thefirm.RectangleFactory;

public class Problem
{
  public static void main(String[] args)
  {
    Rectangle rectangle = RectangleFactory.getInstance();
    rectangle.scale(0);
    System.out.println(rectangle);
  }
}
```

A complaint to Shape Inc.

```
Rectangle rectangle = RectangleFactory.getInstance();
System.out.println(rectangle);
rectangle.scale(0);
System.out.println(rectangle);
```

When the client runs the app, it does not produce the expected output

```
Rectangle of width 0 and height 0
Rectangle of width 0 and height 0
```

Instead, it produces something random like

```
Rectangle of width 0 and height 0
Rectangle of width 148460848 and height 161078995
```

A complaint to Shape Inc.

Question

To determine that there is no bug in the `Problem` app, the APIs of which methods of which classes need to be inspected?

A complaint to Shape Inc.

Question

To determine that there is no bug in the `Problem` app, the APIs of which methods of which classes need to be inspected?

Answer

- The `getInstance` method of the `RectangleFactory` class,
- the `scale` method of the `Rectangle` class, and
- the `toString` method of the `Rectangle` class.

A complaint to Shape Inc.

Question

Is the `getInstance` method of the `RectangleFactory` class used properly?

A complaint to Shape Inc.

Question

Is the `getInstance` method of the `RectangleFactory` class used properly?

Answer

Yes.

A complaint to Shape Inc.

Question

Is the `scale` method of the `Rectangle` class used properly?

A complaint to Shape Inc.

Question

Is the `scale` method of the `Rectangle` class used properly?

Answer

Yes. Note that the precondition is satisfied.

A complaint to Shape Inc.

Question

Is the `toString` method of the `Rectangle` class used properly?

A complaint to Shape Inc.

Question

Is the `toString` method of the `Rectangle` class used properly?

Answer

Yes.

The Rectangle class

Question

Which parts of the `Rectangle` class should be inspected?

The `Rectangle` class

Question

Which parts of the `Rectangle` class should be inspected?

Answer

- The `scale` method and
- the `toString` method.

The Rectangle class

```
package com.quadrangle;

public class Rectangle
{
  ...

  /**
   * Scales this rectangle with the given factor.
   * @param factor the scaling factor.
   * @pre. factor >= 0
   */
  public void scale(int factor)
  {
    this.width = this.width * factor;
    this.height = this.height * factor;
  }
```

The Rectangle class

```
public String toString()
{
  return String.format("Rectangle with width %d and height
                        this.width, this.height);
}
```

The RectangleFactory class

Question

Which parts of the RectangleFactory class should be inspected?

The RectangleFactory class

Question

Which parts of the `RectangleFactory` class should be inspected?

Answer

The `getInstance` method.

The RectangleFactory class

```
import com.quadrangle.Rectangle;

...

   public static Rectangle getInstance()
   {
      return new com.discount.Rectangle();
   }
```

The RectangleFactory class

```
import com.quadrangle.Rectangle;

...

   public static Rectangle getInstance()
   {
      return new com.discount.Rectangle();
   }
```

Question

To which package does the return type of `getInstance` belong?

The RectangleFactory class

```
import com.quadrangle.Rectangle;

...

   public static Rectangle getInstance()
   {
      return new com.discount.Rectangle();
   }
```

Question

To which package does the return type of `getInstance` belong?

Answer

`com.quadrangle`

The RectangleFactory class

```
public static Rectangle getInstance()
{
    return new com.discount.Rectangle();
}
```

The RectangleFactory class

```
public static Rectangle getInstance()
{
    return new com.discount.Rectangle();
}
```

Question

To which package does `new com.discount.Rectangle()` belong?

The RectangleFactory class

```
public static Rectangle getInstance()
{
   return new com.discount.Rectangle();
}
```

Question

To which package does `new com.discount.Rectangle()` belong?

Answer

`com.discount`

The RectangleFactory class

```
public static Rectangle getInstance()
{
    return new com.discount.Rectangle();
}
```

The RectangleFactory class

```
public static Rectangle getInstance()
{
    return new com.discount.Rectangle();
}
```

Question

Although `new com.discount.Rectangle()` belongs to the package `com.discount`, why can the return type of the method belong to the package `com.quadrangle`?

The RectangleFactory class

```
public static Rectangle getInstance()
{
    return new com.discount.Rectangle();
}
```

Question

Although `new com.discount.Rectangle()` belongs to the package `com.discount`, why can the return type of the method belong to the package `com.quadrangle`?

Answer

`com.discount.Rectangle` is a sub class of `com.quadrangle.Rectangle`.

The com.discount.Rectangle class

```
/**
 * Scales this rectangle with the given factor.
 *
 * @param factor the scaling factor.
 * @pre. factor >= 1
 */
public void scale(int factor)
{
  if (factor == 0)
  {
    Random random = new Random(System.currentTimeMillis());
    super.setWidth(Math.abs(random.nextInt()));
    super.setHeight(Math.abs(random.nextInt()));
  }
  else
  {
    super.scale(factor);
  }
}
```

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

Question

During early binding, to which method of which class is the invocation `rectangle.scale(0)` bound?

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

Question

During early binding, to which method of which class is the invocation `rectangle.scale(0)` bound?

Answer

1. The declared type of `rectangle` is `com.quadrangle.Rectangle`.
2. The class `com.quadrangle.Rectangle` contains the following matching methods: `scale(int)`.
3. `scale(int)` is the only and therefore best match.

`scale(int)` of `com.quadrangle.Rectangle`

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

Question

During late binding, to which method of which class is the invocation `rectangle.scale(0)` bound?

The Problem app

```
import com.quadrangle.Rectangle;

Rectangle rectangle = RectangleFactory.getInstance();
rectangle.scale(0);
```

Question

During late binding, to which method of which class is the invocation `rectangle.scale(0)` bound?

Answer

1. The actual type of `rectangle` is `com.discount.Rectangle`.

`scale(int)` of `com.discount.Rectangle`

The Problem app

Question

But if we invoke `scale(int)` of `com.discount.Rectangle` with argument 0, then the precondition is not satisfied. But how could the client know? So, who is to blame?

Question

But if we invoke `scale(int)` of `com.discount.Rectangle` with argument 0, then the precondition is not satisfied. But how could the client know? So, who is to blame?

Answer

The implementer of the class `com.discount.Rectangle` is to blame. When you override a method you cannot strengthen the precondition.

Preconditions

Rule

When you override a method you cannot strengthen the precondition.

If

```
public class A1
{
   /** @pre. P1 */
   public void m(...) { ...}
}

public class A2 extends A1
{
   /** @pre. P2 */
   public void m(...) { ...}
}
```

then P1 $\Rightarrow$ P2.

Preconditions

Since

```
public class com.quadrangle.Rectangle
{
   /** @pre. factor >= 0 */
   public void scale(...) { ...}
}

public class com.discount.Rectangle
  extends com.quadrangle.Rectangle
{
   /** @pre. factor >= 1 */
   public void scale(...) { ...}
}
```

and `factor >= 0` $\not\Rightarrow$ `factor >= 1`, the implementer of the `com.discount.Rectangle` did not follow the rule and is therefore to blame.

Postconditions

Rule

When you override a method you cannot weaken the postcondition.

Exceptions

Rule

If

```
public class A1
{
   public void m(...) throws E1 { ...}
}

public class A2 extends A1
{
   public void m(...) throws E2{ ...}
}
```

then E2 is E1 or a sub class of it.

Chapter 7: Graphical User Interfaces

EECS 1030

moodle.yorku.ca

Graphical user interfaces

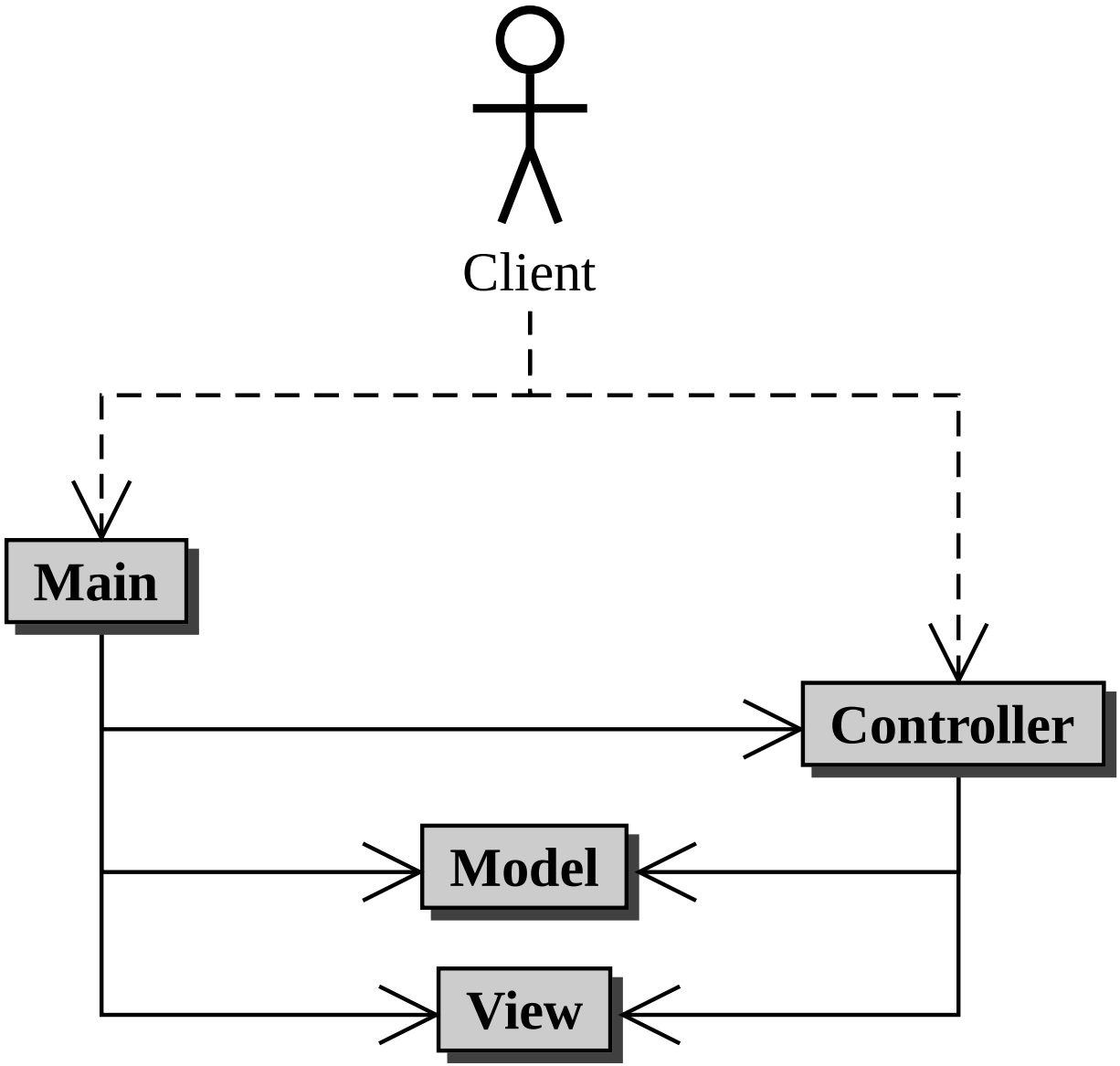
Although we will discuss graphical user interfaces (GUIs), the focus is on aggregation and inheritance.

Model-View-Controller

We use the model-view-controller (MVC) design pattern to implement a GUI.

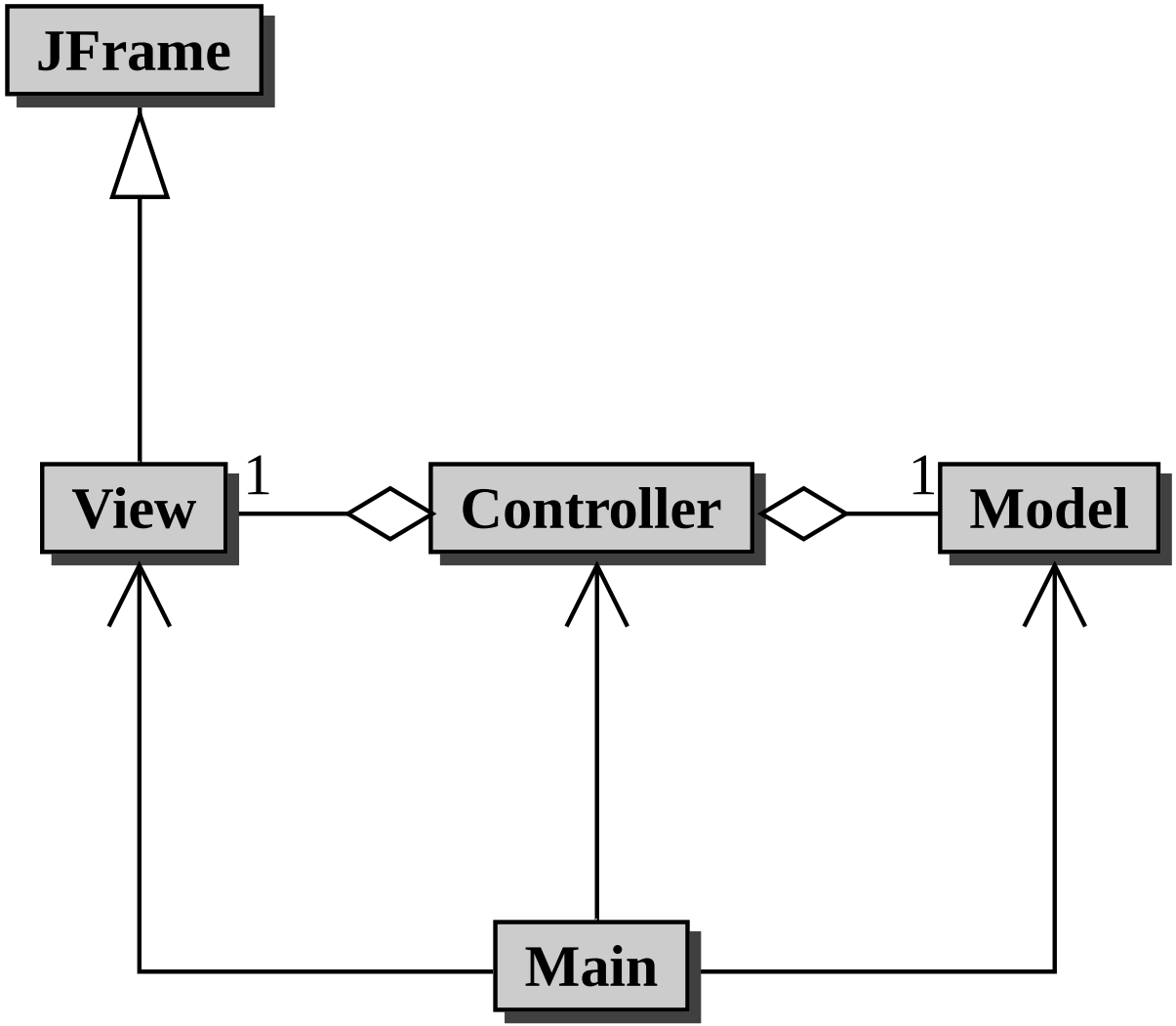
- The model represents the data and provides ways to manipulate the data.
- The view provides a graphical representation of the model.
- The controller translates the client's interactions with the view into actions that manipulate the view and the model. These interactions can be menu selections, button clicks, etc.

Model-View-Controller



Problem

Implement a simple calculator.


JFrame
View
1
Controller
1
Model
Main