Test 3

The second test will be 75 minutes and will consist of two parts.

The programming part will be about Chapter 2-5, excluding Section 2.6, 4.5, 5.2 and 5.3. You will be asked to implement one class. We will already provide you with a skeleton which includes the javadoc. This part will be worth 50% of the marks. If your code does not compile, you get a 50% penalty (that is, your score for the programming part will be divided by 2 if your code does not compile).

The "written" part will also be about Chapter 2-5, excluding Section 2.6, 4.5, 5.2 and 5.3. This part will consist of six questions (two multiple choice, two short answer questions and two longer answer questions). This part will be worth the remaining 50% of the marks.

During the test, you will have access to the textbook. You may bring a blank piece of paper to the test.

Chapter 6: Inheritance

EECS 1030

moodle.yorku.ca

The `extends` keyword

To specify that the `GoldenRectangle` class is a subclass of the `Rectangle` class, we use the following class header:

```
public class GoldenRectangle extends Rectangle
```

Attributes

The private attributes `width` and `height` of the Rectangle class are part of the state of a `GoldenRectangle` object, but are *not* inherited.

As a result, the private attributes `width` and `height` of the `Rectangle` class *cannot* be accessed by their name in the `GoldenRectangle` class.

Constructors

Delegate to a constructor of the `Rectangle` class to initializes the attributes width and height.

Although it may not be the most intuitive syntax, we use

```
super(width, height);
```

`super` has an implicit parameter, namely `this`.

Overridden methods

Delegate to the corresponding method in the super class.

Although it may not be the most intuitive syntax, we use, for example,

```
super.equals(object)
```

`super` has an implicit parameter, namely `this`.

toString method

```
boolean equal;
if (object != null && this.getClass() == object.getClass()
{
   GoldenRectangle other = (GoldenRectangle) object;
   equal = super.equals(other) &&
           this.getWeight() == other.getWeight();
}
else
{
   equal = false;
}
return equal;
```

toString method

Question

Can we simply use

```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

toString method

Question

Can we simply use

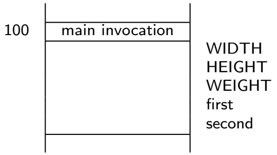
```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

Answer

Yes.

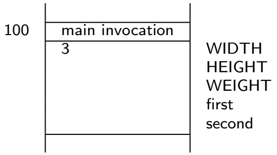
Memory diagram

```
final  int  WIDTH = 3;
final  int  HEIGHT = 6;
final  int  WEIGHT = 80;
GoldenRectangle  first  =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output. println ( first  . equals(second));
```



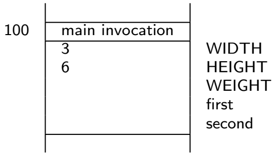
Memory diagram

```
final int WIDTH = 3;
 final  int  HEIGHT = 6;
 final  int  WEIGHT = 80;
GoldenRectangle  first  =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output. println ( first  .equals(second));
```



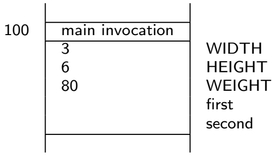
Memory diagram

```
final int WIDTH = 3;
final int HEIGHT = 6;
final int WEIGHT = 80;
GoldenRectangle first =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output.println(first.equals(second));
```



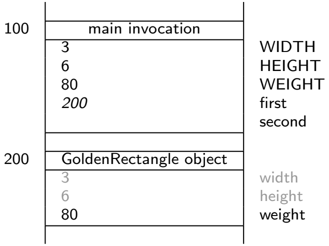
Memory diagram

```
 final  int  WIDTH = 3;
 final  int  HEIGHT = 6;
final int WEIGHT = 80;
GoldenRectangle  first  =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output. println ( first  . equals(second));
```



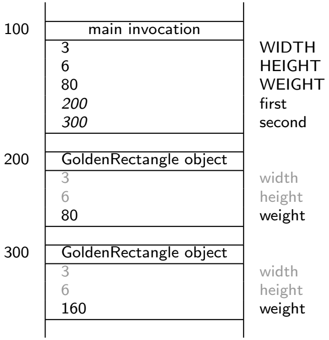
Memory diagram

```
final int WIDTH = 3;
final int HEIGHT = 6;
final int WEIGHT = 80;
GoldenRectangle first =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output.println ( first .equals(second));
```



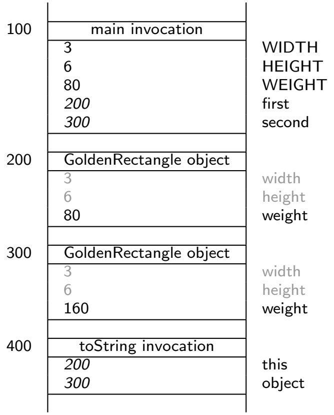
Memory diagram

```
final int WIDTH = 3;
final int HEIGHT = 6;
final int WEIGHT = 80;
GoldenRectangle first =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output.println(first.equals(second));
```



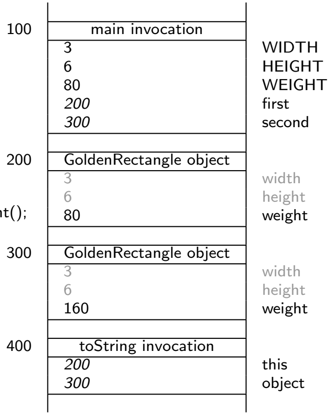
Memory diagram

```
final  int  WIDTH = 3;
final  int  HEIGHT = 6;
final  int  WEIGHT = 80;
GoldenRectangle  first  =
  new GoldenRectangle(WIDTH, HEIGHT, WEIGHT);
GoldenRectangle second =
  new GoldenRectangle(WIDTH, HEIGHT, 2 * WEIGHT);
output. println (first.equals(second));
```



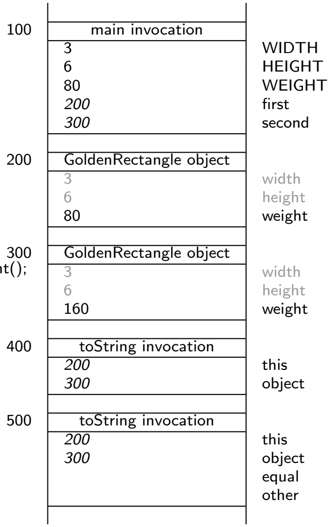
Memory diagram

```
return super.equals(object) &&
  this.getWeight() == ((GoldenRectangle) object).getWeight();
```



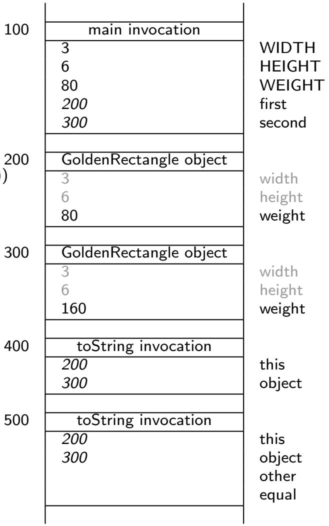
Memory diagram

```
return  super.equals(object) &&
  this .getWeight() == ((GoldenRectangle) object).getWeight();
```



Memory diagram

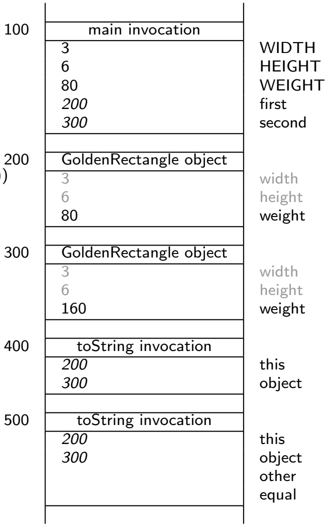
```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



100	main invocation	
	3	WIDTH
	6	HEIGHT
	80	WEIGHT
	200	first
	300	second
200	GoldenRectangle object	
	3	width
	6	height
	80	weight
300	GoldenRectangle object	
	3	width
	6	height
	160	weight
400	toString invocation	
	200	this
	300	object
500	toString invocation	
	200	this
	300	object
		other
		equal

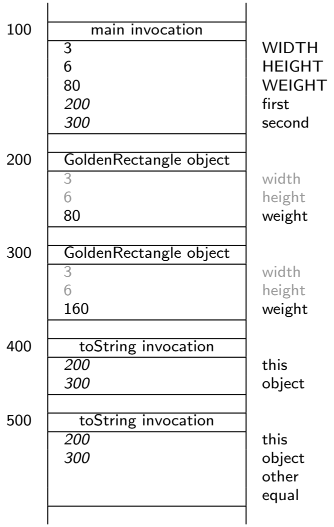
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



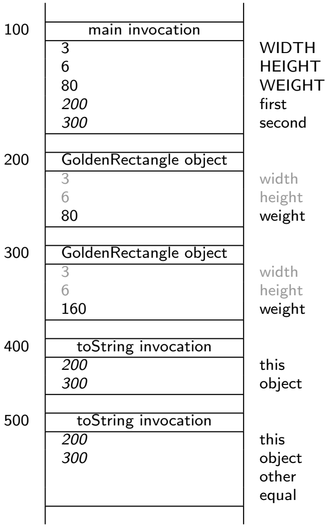
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth &&
            this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



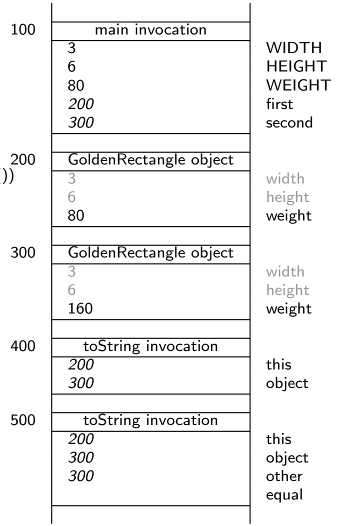
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



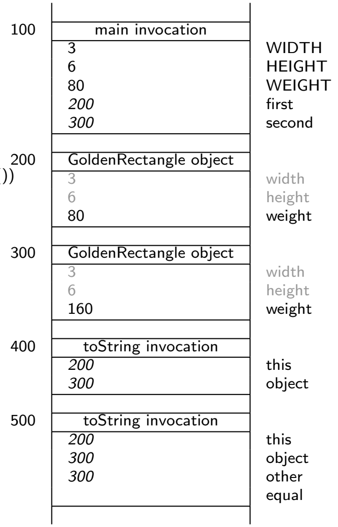
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



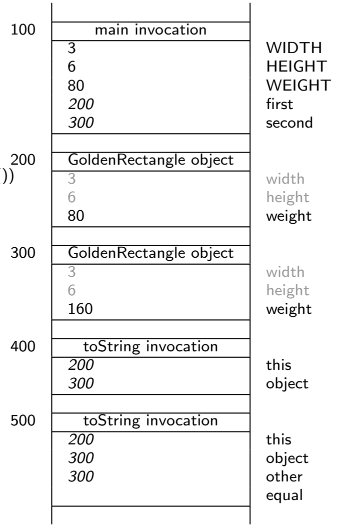
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth() &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



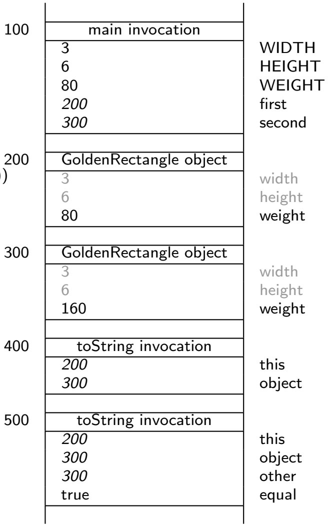
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth() &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



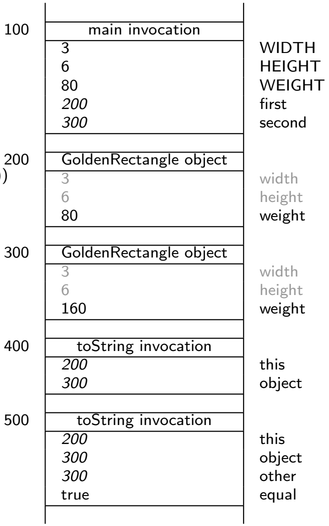
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth() &&
           this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



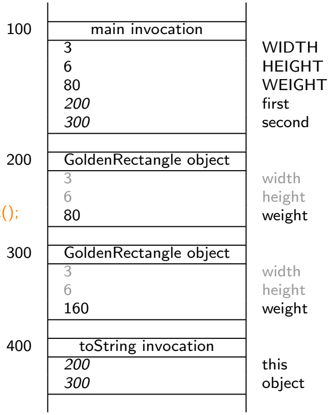
Memory diagram

```
boolean equal;
if (object != null && this.getClass() == object.getClass())
{
   Rectangle other = (Rectangle) object;
   equal = this.getWidth() == other.getWidth() &&
            this.getHeight() == other.getHeight();
}
else
{
   equal = false;
}
return equal;
```



Memory diagram

```
return super.equals(object) &&
  this.getWeight() == ((GoldenRectangle) object).getWeight();
```



toString method

```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

Question

What happens when `object` is `null`?

toString method

```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

Question

What happens when `object` is `null`?

Answer

`super.equals(object)` returns `false` and therefore `this.getWeight() == ((GoldenRectangle) object).getWeight()` is not executed (so no `NullPointerException`).

toString method

```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

Question

What happens when `object` is not a `GoldenRectangle`?

toString method

```
return super.equals(object) &&
       this.getWeight() == ((GoldenRectangle) object).getWe
```

Question

What happens when `object` is not a `GoldenRectangle`?

Answer

`super.equals(object)` returns `false` and therefore `this.getWeight() == ((GoldenRectangle) object).getWeight()` is not executed (so no `ClassCastException`).

Implement your own Exception class

Problem

Implement the `PricingException` class, the API of which can be found here.

Implement your own Exception class

Question

What is the class header?

Implement your own Exception class

Question

What is the class header?

Answer

```
public class PricingException extends Exception
```

Implement your own Exception class

Question

Which attributes are part of the state of a `PricingException` object?

Implement your own Exception class

Question

Which attributes are part of the state of a `PricingException` object?

Answer

An attribute named `message` of type `String`.

Implement your own Exception class

Question

Which attributes are part of the state of a `PricingException` object?

Answer

An attribute named `message` of type `String`.

Question

Do we have to declare this attribute in the `PricingException` class?

Implement your own Exception class

Question

Which attributes are part of the state of a `PricingException` object?

Answer

An attribute named `message` of type `String`.

Question

Do we have to declare this attribute in the `PricingException` class?

Answer

No, because it is already present in the super class `Throwable`.

Implement your own Exception class

Problem

Implement the constructors.

Combining inheritance and aggregation

Problem

Implement the `ColouredRectangle` class, the API of which can be found here.

Combining inheritance and aggregation

Question

What is the class header?

Combining inheritance and aggregation

Question

What is the class header?

Answer

```
public class ColouredRectangle extends Rectangle
```

Combining inheritance and aggregation

Question

Which attributes are part of the state of a `ColouredRectangle` object?

Combining inheritance and aggregation

Question

Which attributes are part of the state of a `ColouredRectangle` object?

Answer

The attributes `width` and `height` of type `int` and the attribute `colour` of type `Color`.

Combining inheritance and aggregation

Question

Which attributes are part of the state of a `ColouredRectangle` object?

Answer

The attributes `width` and `height` of type `int` and the attribute `colour` of type `Color`.

Question

Which do we have to declare in the `ColouredRectangle` class?

Combining inheritance and aggregation

Question

Which attributes are part of the state of a `ColouredRectangle` object?

Answer

The attributes `width` and `height` of type `int` and the attribute `colour` of type `Color`.

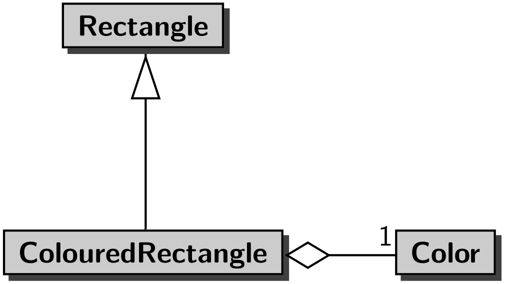
Question

Which do we have to declare in the `ColouredRectangle` class?

Answer

Only the attribute `colour` of type `Color`.

Combining inheritance and aggregation



Combining inheritance and aggregation

Question

When implementing the constructors, how do we delegate?

Combining inheritance and aggregation

Question

When implementing the constructors, how do we delegate?

Answer

This can be done in different ways. For example, the copy constructor delegates to the three-parameter constructor, and the three-parameter constructor delegates to a constructor of the super class.