Test 2

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

The programming part will be about Chapter 2, excluding Section 2.6. You will be asked to implement a utility class with four methods (an easy one, two of medium difficulty, and a challenging one). You do not have to include javadoc in your code. This part will be worth 70% 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 be about Chapter 3, excluding Section 3.3. This part will consist of three questions (one multiple choice, one short answer question and one longer answer question). This part will be worth the remaining 30% of the marks.

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

Static attributes and methods

In general, static attributes capture data that is associated with the class, not with individual objects.

In general, static methods manipulate the passed arguments and static attributes.

Static attributes and methods

Question

How do you get the value of the static attribute `CENTIMETER` of the class `Converter`?

Static attributes and methods

Question

How do you get the value of the static attribute `CENTIMETER` of the class `Converter`?

Answer

```
Converter.CENTIMETER
```

Static attributes and methods

Question

How do you get the value of the static attribute `CENTIMETER` of the class `Converter`?

Answer

`Converter.CENTIMETER`

Question

How do you invoke the static method `convert` of the class `Converter` with the arguments `1`, `Converter.METER` and `Converter.CENTIMETER`?

Static attributes and methods

Question

How do you get the value of the static attribute `CENTIMETER` of the class `Converter`?

Answer

```
Converter.CENTIMETER
```

Question

How do you invoke the static method `convert` of the class `Converter` with the arguments `1`, `Converter.METER` and `Converter.CENTIMETER`?

Answer

```
Converter.convert(1,
                  Converter.METER,
                  Converter.CENTIMETER)
```

Non-static attributes and methods

In general, non-static attributes capture data that is associated with individual objects.

In general, non-static methods and constructors manipulate the passed arguments and non-static attributes (but may also manipulate static attributes as we will see later in this lecture).

Non-static attributes and methods

Question

How do you get the value of the non-static attribute `width` of the object `rectangle`?

Non-static attributes and methods

Question

How do you get the value of the non-static attribute `width` of the object `rectangle`?

Answer

`rectangle.width` (however, usually the non-static attributes are private and therefore one would have to use `rectangle.getWidth()`).

Non-static attributes and methods

Question

How do you get the value of the non-static attribute `width` of the object `rectangle`?

Answer

`rectangle.width` (however, usually the non-static attributes are private and therefore one would have to use `rectangle.getWidth()`).

Question

How do you invoke the non-static method `scale` on the object `rectangle` with the argument 2?

Non-static attributes and methods

Question

How do you get the value of the non-static attribute `width` of the object `rectangle`?

Answer

`rectangle.width` (however, usually the non-static attributes are private and therefore one would have to use `rectangle.getWidth()`).

Question

How do you invoke the non-static method `scale` on the object `rectangle` with the argument 2?

Answer

```
rectangle.scale(2)
```

Static versus non-static

- static features: class
- non-static features : object

I will try to always associate

- static features with a class (name)
- non-static features with an object (reference)

Chapter 4: Mixing Static and Non-Static Features

EECS 1030

moodle.yorku.ca

Counting

Problem

Modify the `Rectangle` class so that we keep track of the number of `Rectangle` objects that have been created.

Counting: specification

Question

What do we add to the `Rectangle` API?

Counting: specification

Question

What do we add to the `Rectangle` API?

Answer

A method that returns the number of created `Rectangle` objects.

Counting: specification

Question

What do we add to the `Rectangle` API?

Answer

A method that returns the number of created `Rectangle` objects.

Question

Is this method static or non-static?

Counting: specification

Question

What do we add to the `Rectangle` API?

Answer

A method that returns the number of created `Rectangle` objects.

Question

Is this method static or non-static?

Answer

Static, since it returns information about the class, not about an individual object.

Counting: design

Question

`public static int getNumber()`

We need to store the number of created `Rectangle` objects in an attribute. Is this attribute static or non-static?

Counting: design

Question

`public static int getNumber()`

We need to store the number of created `Rectangle` objects in an attribute. Is this attribute static or non-static?

Answer

Static, since it contains information about the class, not about an individual object.

Counting: design

Question

`public static int number`

Where do we need to increment this number?

Counting: design

Question

`public static int number`

Where do we need to increment this number?

Answer

In the constructor.

Counting: implementation

Question

Where do we initialize the attribute `number`?

Counting: implementation

Question

Where do we initialize the attribute `number`?

Answer

Where is it declared, since it is a static attribute.

Counting: implementation

Question

Where do we initialize the attribute `number`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment the attribute `number` in the constructor?

Counting: implementation

Question

Where do we initialize the attribute `number`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment the attribute `number` in the constructor?

Answer

`Rectangle.number++` (don't forget the class name)

Counting: implementation

Question

Where do we initialize the attribute `number`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment the attribute `number` in the constructor?

Answer

`Rectangle.number++` (don't forget the class name)

Question

How do we return the value of the attribute `number` in `getNumber`?

Counting: implementation

Question

Where do we initialize the attribute `number`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment the attribute `number` in the constructor?

Answer

`Rectangle.number++` (don't forget the class name)

Question

How do we return the value of the attribute `number` in `getNumber`?

Answer

`return Rectangle.number` (don't forget the class name)

Contact

Problem

Implement this API of the `Contact` class.

Client of Contact

Problem

Write an app that creates contacts with names "You" and "Me" and addresses "Here" and "There".

Client of Contact

Problem

Draw the memory diagram that represents memory when the execution reaches the end of the `main` method.

Contact revisited

Problem

Modify the `Contact` class such that each `Contact` object has a unique number. The first contact has number 10001, the second one 10002, etc

Contact revisited

Question

How do we represent the unique number of each `Contact` object in the `Contact` class?

Contact revisited

Question

How do we represent the unique number of each `Contact` object in the `Contact` class?

Answer

As a non-static attribute.

Contact revisited

Question

How do we represent the unique number of each `Contact` object in the `Contact` class?

Answer

As a non-static attribute.

Question

What is the type of the attribute?

Contact revisited

Question

How do we represent the unique number of each `Contact` object in the `Contact` class?

Answer

As a non-static attribute.

Question

What is the type of the attribute?

Answer

`int`

Contact revisited

Problem

Add the non-static attribute and its public accessor and private mutator.

Contact revisited

Problem

Add the non-static attribute and its public accessor and private mutator.

Problem

Add the new non-static attribute to the memory diagram.

Contact revisited

Question

Where do we initialize the new non-static attribute?

Contact revisited

Question

Where do we initialize the new non-static attribute?

Answer

In the constructor.

Contact revisited

Question

If we add a parameter to the constructor, can a client create two `Contact` objects with the same number?

Contact revisited

Question

If we add a parameter to the constructor, can a client create two `Contact` objects with the same number?

Answer

Yes.

Contact revisited

Question

If we add a parameter to the constructor, can a client create two `Contact` objects with the same number?

Answer

Yes.

Conclusion

Since the number has to be unique, we cannot add a parameter to the constructor.

Contact revisited

```
public Contact(String name, String address)
{
    this.setName(name);
    this.setAddress(address);
    this.setNumber(?);
}
```

Question

The first time the constructor is invoked, which value should be assigned to `number`?

Contact revisited

```
public Contact(String name, String address)
{
    this.setName(name);
    this.setAddress(address);
    this.setNumber(?);
}
```

Question

The first time the constructor is invoked, which value should be assigned to `number`?

Answer

10001

Contact revisited

```
public Contact(String name, String address)
{
    this.setName(name);
    this.setAddress(address);
    this.setNumber(?);
}
```

Question

The second time the constructor is invoked, which value should be assigned to `number`?

Contact revisited

```
public Contact(String name, String address)
{
    this.setName(name);
    this.setAddress(address);
    this.setNumber(?);
}
```

Question

The second time the constructor is invoked, which value should be assigned to `number`?

Answer

10002

Contact revisited

```
public Contact(String name, String address)
{
    this.setName(name);
    this.setAddress(address);
    this.setNumber(?);
}
```

Question

The second time the constructor is invoked, which value should be assigned to `number`?

Answer

10002

Observation

This is very similar to the counting of the number of created `Rectangle` objects.

Contact revisited

Question

```
public static int nextNumber
```

Where do we need to increment this number?

Contact revisited

Question

`public static int nextNumber`

Where do we need to increment this number?

Answer

In the constructor.

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Answer

Where is it declared, since it is a static attribute.

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment `nextNumber` in the constructor?

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment `nextNumber` in the constructor?

Answer

`Contact.nextNumber++` (don't forget the class name)

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment `nextNumber` in the constructor?

Answer

`Contact.nextNumber++` (don't forget the class name)

Question

How do we initialize the attribute `nextNumber`?

Counting: implementation

Question

Where do we initialize the attribute `nextNumber`?

Answer

Where is it declared, since it is a static attribute.

Question

How do we increment `nextNumber` in the constructor?

Answer

`Contact.nextNumber++` (don't forget the class name)

Question

How do we initialize the attribute `nextNumber`?

Answer

```
public static int nextNumber = Contact.FIRST_NUMBER;
```

equals method

Question

```
if (object != null && this.getClass() == object.getClass())
{
    Contact other = (Contact) object;
    equal = this.getName().equals(other.getName()) &&
            this.getAddress().equals(other.getAddress());
}
```

If the name or address of this Contact is null, what happens?

equals method

Question

```
if (object != null && this.getClass() == object.getClass())
{
    Contact other = (Contact) object;
    equal = this.getName().equals(other.getName()) &&
            this.getAddress().equals(other.getAddress());
}
```

If the name or address of this Contact is null, what happens?

Answer

A NullPointerException is thrown.

equals method

Question

```
if (object != null && this.getClass() == object.getClass())
{
    Contact other = (Contact) object;
    equal = this.getName().equals(other.getName()) &&
            this.getAddress().equals(other.getAddress());
}
```

If the `name` or `address` of `this Contact` is `null`, what happens?

Answer

A `NullPointerException` is thrown.

Claim

If this ever happens, it is not the implementer's fault. But how do we prove that it is not the implementer's responsibility?

Responsibility

Question

How can we argue that the attributes `name` and `address` are never `null`? Which parts of the code do we need to consider?

Responsibility

Question

How can we argue that the attributes `name` and `address` are never `null`? Which parts of the code do we need to consider?

Answer

Those parts that assign values to the attributes `name` and `address`:

- the constructor
- the mutators

Responsibility

Question

Can the constructor assign `null` to either `name` or `address`?

Responsibility

Question

Can the constructor assign `null` to either `name` or `address`?

Answer

Yes.

Responsibility

Question

Can the constructor assign `null` to either `name` or `address`?

Answer

Yes.

Question

If this would cause a crash, would the implementer be responsible?

Responsibility

Question

Can the constructor assign `null` to either `name` or `address`?

Answer

Yes.

Question

If this would cause a crash, would the implementer be responsible?

Answer

No, because the precondition states `number != null, address != null`.

Responsibility

Question

Assuming that the client satisfies the preconditions, can the mutators assign `null` to either `name` or `address`?

Responsibility

Question

Assuming that the client satisfies the preconditions, can the mutators assign `null` to either `name` or `address`?

Answer

No, because the mutators are used properly in the constructor and they are private (so they cannot be invoked by the client).

Class invariant

We have just shown that `this.number != null && this.address != null` is a class invariant.

Definition

A class invariant is a Boolean expression with the following two properties:

- It is true after each public constructor invocation, provided that the client ensures that the precondition of the invoked constructor is met.
- It is maintained by each public method invocation, provided that the client ensures that the precondition of the invoked method is met.

Class invariant

I will document a class invariant as

```
//@ invariant this.number != null && this.address != null
```

Class invariant

Question

What do we need to check to conclude `this.number != null && this.address != null` is a class invariant?

Class invariant

Question

What do we need to check to conclude `this.number != null && this.address != null` is a class invariant?

Answer

If `number != null` and `address != null` (precondition) then after the execution of the constructor

```
this.setName(name);
this.setAddress(address);
```

we have that `this.number != null && this.address != null`.

Class invariant

Question

What else do we need to check to conclude `this.number != null && this.address != null` is a class invariant?

Class invariant

Question

What else do we need to check to conclude `this.number != null && this.address != null` is a class invariant?

Answer

If `this.number != null && this.address != null` holds before the execution of `getName`, `getAddress`, `equals`, `hashCode` and `toString`, then `this.number != null && this.address != null` also holds after the execution of those methods.