Question 1

1 / 1 pts

The `assertEquals` method is **not** useful for comparing values of type...

- `double`
- `int`
- `String`
- `int[]` (selected)

Question 2

1 / 1 pts

Sometimes we put the tester code and the tested code under two different base folders. What is the main reason for that?

- the tester code can use different classes
- the tester code can use different packages
- to avoid packaging the tester code and the tested code together (selected)
- to avoid compiling the tester code and the tested code together

Incorrect

Question 3

0 / 1 pts

The `hu.finance.bank.transaction.test.TransactionTest` class is used for testing which class?

- `hu.finance.bank.transaction.Transaction`
- ◉ `hu.finance.bank.transaction.test.Transaction`
- any class in the package `hu.finance.bank.transaction`
- (no restriction)

Incorrect

Question 4

0 / 1 pts

Which step do we **not** use to run JUnit 4 tests?

- put `import static org.junit.Assert.*;` in the code
- ◉ add the `.jar` file of the `hamcrest-core` library to the classpath
- put `import static org.junit.JUnit.*;` in the code
- add the `.jar` file of the `junit` library to the classpath

Question 5

1 / 1 pts

Which of the following is **bad practice** using JUnit?

- ◉ to put all `assert...()` calls in the same method
- to put the `@Test` annotation on all methods of a class
- to put all `assert...()` calls in separate methods
- to put the `@Test` annotation only on those methods of a class whose name begin with `test`

Question 1

1 / 1 pts

Let us suppose that `e1.hashCode() == e2.hashCode()`. Into a `HashSet`, we first add `e1` then `e2`. Can both elements be in the set afterwards?

- (●) yes, if `e1.equals(e2)` is true
- ( ) yes, always
- ( ) yes, if `e1.equals(e2)` is false
- ( ) no, never

Incorrect

Question 2

0 / 1 pts

If our class is defined as `class MyClass {...}`, what is it equivalent to?

- ( ) `class MyClass extends java.lang.Object {...}`
- ( ) `class MyClass extends java.util.Object {...}`
- (●) `class MyClass implements java.util.Object {...}`
- ( ) `class MyClass implements java.lang.Object {...}`

Incorrect

Question 3

0 / 1 pts

We define `class WhatHappens { /* empty class body */ }` and create two instances in the variables `wh1` and `wh2`. What does `wh1.equals(wh2)` evaluate to?

- ( ) `true`
- ( ) `false`
- ( ) an exception is thrown
- (●) depending on some condition, it can be either `true` or `false`

Incorrect

Question 4

0 / 1 pts

Let `arr` be an `ArrayList<Integer>` and `lili` be a `LinkedList<Integer>`. Both contain a single element `"a"`. Is it true that `List.of("a").equals(arr)`? Is it true that `lili.equals(arr)`?

- ( ) false, true
- (●) false, false
- ( ) true, false
- ( ) true, true

Question 5

1 / 1 pts

Which method is **not** in the class `Object`?

- `equals`
- `toString`
- `hashCode`
- `println`

Week 7

Incorrect

Question 1

0 / 1 pts

What do we get if we invoke `"123".compareTo("123")`

- `true`
- some other `int` value
- `123`
- `false`
- compilation error

Incorrect

Question 2

0 / 1 pts

What is type erasure?

- the type parameter of a generic type is not stored in the `.java` file
- the type parameter of a generic type is not stored in the `.class` file
- the type parameter of a generic type is computed during execution
- the type parameter of a generic type is computed during compilation

Incorrect

Question 3

0 / 1 pts

Out of the following, what is the (static) type of `getElemsOutOfMe` in `for (List<String> elem : getElemsOutOfMe) {...}`?

- (•) `Iterator<List<String>>`
- ( ) `Comparator<List<String>>`
- ( ) `Iterable<List<String>>`
- ( ) `Comparable<List<String>>`

Incorrect

Question 4

0 / 1 pts

Which of the following does **not necessarily** have a side effect?

- ( ) reading data from a database
- ( ) printing data to the standard output
- ( ) setting the value of a local variable
- (•) setting the value of a static variable

Question 5

1 / 1 pts

What is a typical operation that makes use of `Comparable`?

- ( ) `toString`
- (•) sorting
- ( ) `hashCode`
- ( ) printing

Week 8

Incorrect

Question 1

0 / 2 pts

What is “scope”?

- ○ it’s about the presence of a variable in memory at runtime
- ○ it’s about the visibility of a variable at compile time
- ○ it’s about the presence of an object in memory at compile time
- ◉ it’s about the visibility of a variable at runtime
- ○ it’s about the presence of an object in memory at runtime
- ○ it’s about the presence of a variable in memory at compile time
- ○ it’s about the visibility of an object at compile time
- ○ it’s about the visibility of an object at runtime

Incorrect

Question 2

0 / 2 pts

In Java, what can hide what?

- ☐ a local variable’s name can hide a field’s name
- ☑ a local variable’s name can hide a method parameter’s name
- ☑ a field’s name can hide a method parameter’s name
- ☐ a field’s name can hide a local variable’s name
- ☐ a method parameter’s name can hide a local variable’s name
- ☐ a method parameter’s name can hide a field’s name

Question 3

1 / 1 pts

When should you make an overriden version of `finalize`?

- ○ if `main` contains a `try` with a `finally` clause
- ○ if you override `toString`, too
- ○ if you create a `final` class
- ◉ never

Theory exam

Question 1

Not yet graded / 0 pts

If you have missed any quizzes due to some legitimate reason (e.g. you were registered late for the course by the admins or you caught COVID), insert the number of weeks that you've missed on the first line of this "answer", and on the second line, write the details (which weeks you've missed and why). If you haven't missed any quizzes at all (at least not without a proper reason), **leave this "answer" empty**.

Your Answer:

I missed 3 quizzes because I have over lab.

sorry but I can't put the dates because in this case I need to leave the quiz 🙃.

Question 2

Not yet graded / 5 pts

We have an `ExecuteMe.java` that is compilable and runnable under the folder structure `this/is/my/folder/structure`. Based on this much information, what can you tell me about the source code in the file, and the compilation and execution of the program?

Your Answer:

ExecuteMe.java compilable and runnable under the folder so for this java file we don't have any access problem package well written and well access ligne, but we should put the package on all the other file and it's better if we use test folder like this all the compilation and execution will be okay. and we need to create getters and setters for any fields with private access and throw some exceptions to grants that we will not get compilation problem.

if we applied those roles we will get a very good structure and easy way of coding and testing.

Question 3

Not yet graded / 5 pts

Why and how does aliasing occur in every program (except small, insignificant ones) during execution?

Your Answer:

we have aliasing when we have more then one reference for the same object it's like over reference for the object when it will appear and not expected so in this case a compilation error will appear.

Question 4

Not yet graded / 5 pts

We change our source code which was valid at the beginning: in one of our classes, we replace the `class` keyword as `interface`, and we change the bodies of its methods to a semicolon (`;`). Every other part of the source code is left as it was. What sorts of compilation errors can be expected? Can it happen that our code compiles and runs all right after the change?

Your Answer:

so if the previous class has fields so in this case we need to rwmove the fieldes because we are not allowed to change "class" by "interface" because interfaces contains just methods and also we need to remove the contractors if they appeared or we will fake a compilation error then if we remove (;) from the function body. and even we left everything empty and compile it will compile. but we have to make sure that we remove the class file from the folder because we are not allowed to have ! class and interface with the same name

Question 5

Not yet graded / 5 pts

The `Product` class has a textual `name` field and a `price` which is an `int`. We have two sets of `Product` objects: pairwise, their fields contain the same data. If we put these objects into a) two lists, b) two arrays, and c) a list and an array, and then we invoke the standard equality checking method on them, what will be the result in each case and why?

Your Answer:

a) two lists :

False because they have different memory adresse.

b) two array:

False because they have different memory adresse.

3) a list and an array:

False because they have different memory adresse.

PS : if we compare the references of the objects we will get true because the both are int and has same values

LAB

Incorrect

Question 1

0 / 0.5 pts

What is the default value of an instance variable?

- args
- null
- it depends on the variable type
- (x) it has no value
- 0

Question 2

0.5 / 0.5 pts

To declare an array in Java, the variable type should be defined with:

- arr
- (x) []
- ()
- {} and []
- {}

Question 1

1 / 1 pts

```
enum Colors
{
   private GREEN,
   public RED,
   BLUE;
}
```

what is wrong in the above program?

- nothing wrong in it!
- we should use Class instead of enum!
- **You can't declare enum constants with any modifiers. They are public, static and final by default.**
- enum constants should be written in small letters.
- BLUE should be public too.

Question 1

1 / 1 pts

file1:MySet.java

```
---------------------
import java.util.HashSet;
import java.util.Set;


public class MySet<E>{
   private final Set<E> set = new HashSet<>();
   public void add (E element) throws AnException {
      if (contains(element)) {
         throw new AlreadyContainedException();
      }
      set.add(element);
   }
}
file2:AnException.java
---------------------------
public class AnException  {
```


```
---------------------------
public class AnException  {

}
```


There is one error in the files, what is it?

- set.add(element); is invalid expression!
- Correct! AnException should extends Exception class.
- E should be replaced by a wrapper class type.
- we should not import java.util.Set;

Question 1

1 / 1 pts

```
package c;
import a.b.D;


public class C
{
   private double x = 0;
   private double y = 0;
   private double r = 1;


   public C(double x, double y, double r)
   {
      this.x = x;
      this.y = y;
      if (r <= 0)
      {
         throw new IllegalArgumentException();
      }
      this.r = r;
```

```
        this.r = r;
    }
    public double getX() { return x; }
    public double getY() { return y; }
    public double getR() { return r; }
}
```

for compiling the above program we do one of the following:

- javac c/c.java
- javac c/C.java
- javac a/b/D.java
- javac c/a.b.C.java
- javac c/a.b.D.java
- javac C/c.java
- javac C/C.java

Question 1

0.5 / 0.5 pts

we can put multiple classes in the same java file.

- (●) True
- ( ) False

Question 2

0.5 / 0.5 pts

What will happen when you try to compile a java class for which we did not define a constructor?

- ( ) we dont need to compile such file.
- ( ) it depends on the program.
- (●) nothing will happen, it will compile normally.
- ( ) it will give a compilation error.

Question 1

0.5 / 0.5 pts

```
@Test
  public void aTest() {

     assertEquals(null, Book.make(12, "def", "BOOK", 2.34));
  }
```

From the first look, The above tester's result expectation is IllegalArgumentException().

- ○ True
- Correct! ◉ False

Question 2

0.5 / 0.5 pts

JUnit provides Test runners for running tests.

- Correct! ◉ True
- ○ False

Quiz Score: **1** out of 1

Rahmani Manel (DDPNIX) submitted Nov 19, 2021 at 12:16pm

Score for this quiz: **0.5** out of 1
Submitted Nov 19, 2021 at 12:16pm
This attempt took 1 minute.

Question 1

0 / 0.5 pts

Which types can be generic parameters?

Correct Answer

- non-primitive types

You Answered

- primitive types

- all types

Question 2

0.5 / 0.5 pts

What are generic methods?

- They are methods that take void parameters.
- They are methods that extend a generic class.

Correct!

- They are methods that introduce their own type parameters.

Quiz Score: **0.5** out of 1

Question 1

0.5 / 0.5 pts

If the method is protected, then ...

- It is accessible from the subclasses only
- It is accessible from inside the class only
- It is accessible from everywhere.

Correct!

- (selected) It is accessible from inside the class and the subclasses

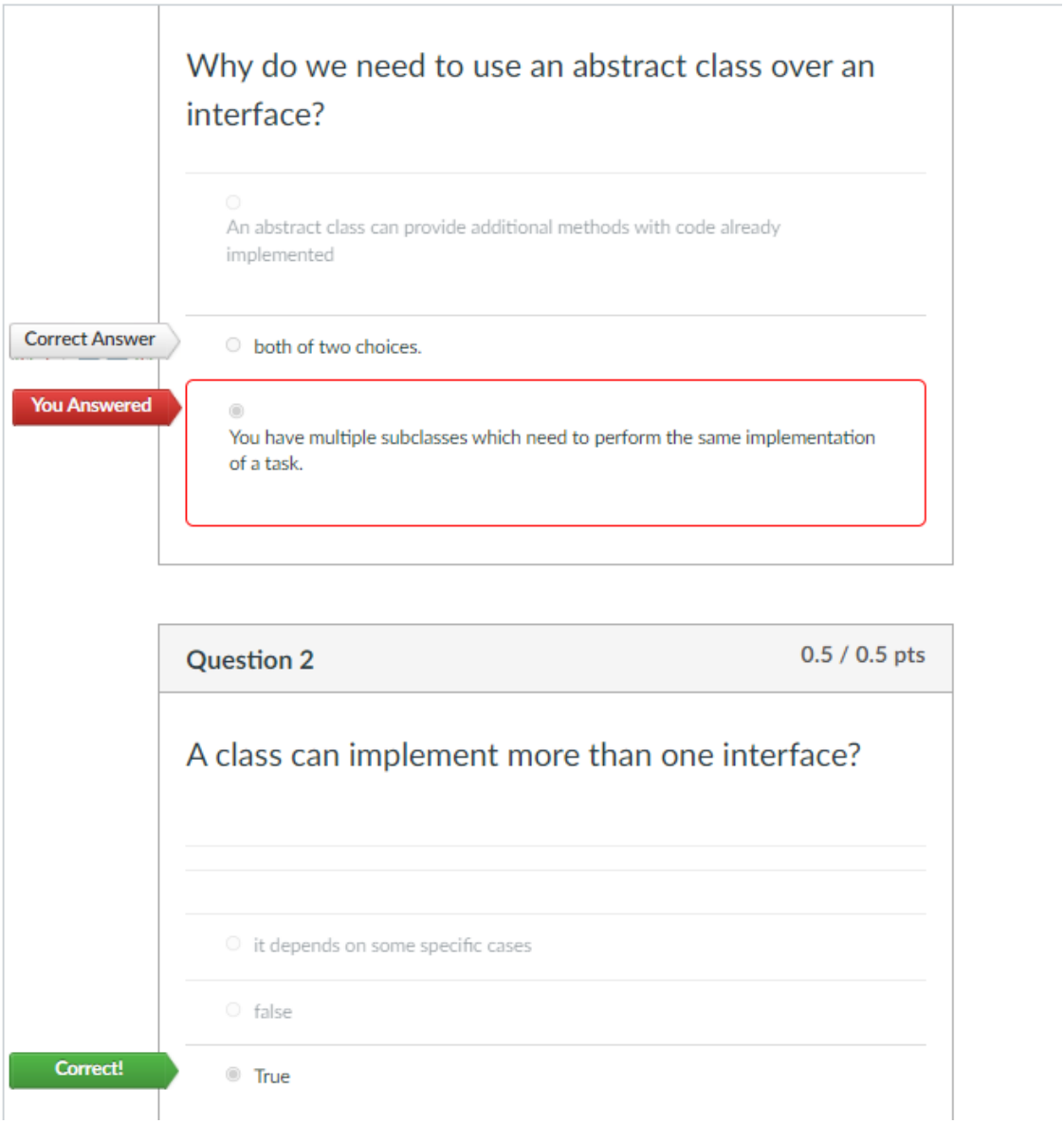
Question 2

0.5 / 0.5 pts

Which of the following statements is false regarding 'finally' block in Java?

Correct!

- (selected) There can be multiple 'finally' blocks for a try block
- Before the termination of the program, JVM executes the 'finally' block (if any).
- The 'finally' block can be used for writing the cleanup code.


Why do we need to use an abstract class over an interface?
An abstract class can provide additional methods with code already implemented
Correct Answer
both of two choices.
You Answered
You have multiple subclasses which need to perform the same implementation of a task.
Question 2
0.5 / 0.5 pts
A class can implement more than one interface?
it depends on some specific cases
false
Correct!
True

LAB exam 😊

Programming Languages Exam 2021-12-22 Lab

Due No due date **Points** 40 **Questions** 1
Available Dec 22, 2021 at 5pm - Dec 22, 2021 at 8pm about 3 hours **Time Limit** None

Instructions

About the exam

Remember that the conditions for the practical exams (see the file about them in the theoretical Canvas) apply.

- Specifically, don't forget to keep sending in your Code Together links using the assignment in Canvas.
- Also remember what you need to do at the end of the exam (the details of solution submission).

You may download the JUnit libraries required for testing by clicking here .

- In this exam we provide the JUnit tests for you. Keep the tester code unmodified except for the following.
  - You may temporarily comment out the yet unimplemented parts.
  - It is part of your task to implement the TODO bits in the tester code.
- Run the tests like this. (Under Linux, use `:` instead of `;`)

```
javac -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." tests\TravelTest.java
java -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." org.junit.runner.JUnitCore tests.TravelTest
```

Task 1: DateAndTime, Destination, DestinationUtils, Flight (14 Points)

Create the `exam20211222.travel.DateAndTime` class with the following fields that are not accessible from other classes: `year`, `month`, `day`, `hour`, and `minute` all of type `int`. All of these fields have getters.

- The class has two constructors:
  1. A constructor that takes all the fields, and sets their values accordingly.
  2. A constructor that takes no parameters. It calls the previously defined constructor with the following values:
     - `year`: 2021
     - `month`: 12
     - `day`: 22
     - `hour`: 04
     - `minute`: 30
  - Hint: Calling an other constructor can be done with the `this(parameters)`.
- Create an implementation the standard method that creates the textual representation of a `DateAndTime` object returns a text like this: `2021.12.22 at 4:30`
- Create `getTime()` which returns a text like this: `4:30`

Create an enum `exam20211222.travel.Destination` which includes five destinations: `BERLIN`, `ROME`, `AMSTERDAM`, `PARIS`, and `HELSINKI`.

Create a class `exam20211222.travel.DestinationUtils`.

The methods in `DestinationUtils` are the following:

- Its static method `getDestination()` takes a `String` argument which represents a travel duration. The string is formatted as `hour:minute` (you **don't** have to check if it's formatted properly); the method returns the proper `exam20211222.travel.Destination` value based on the following durations:

```
BERLIN      01:34
ROME        01:45
AMSTERDAM   02:05
PARIS       02:20
HELSINKI    02:43
```

    - For example, the return value of `getDestination("01:45")` is `Destination.ROME`.
    - If the duration is not listed above, the method returns `null`.
        - Note that it is very bad to return `null`, and you shouldn't do it in real code.
- Its static method `getDestinationDuration()` takes a `Destination` instance and returns the proper duration as string from the table above.
- Its static method `getRoundedHours()` takes a `Destination` instance and returns the number of hours in the duration associated to the argument, plus one more if it contains at least `30` minutes.
    - Hint: partition the duration text along the `:` character returned by `getDestinationDuration()`, then convert the two parts to integers.

Create an `exam20211222.travel.flying.Flight` class with the following protected fields: `name` of type String, `destinationCity` of type `Destination`, `numberOfTravellers` of type int, and `flightDateAndTime` of type `DateAndTime`.

- Create getters for all of the fields, the `destinationCity` getter should return the name of the constant, while the `flightDateAndTime` getter should return its textual representation.
- `Flight` has two constructors:
    1. A constructor that takes all the fields, and sets their values accordingly.
        - Before doing so, check whether `numberOfTravellers` is at least `15` and at most `100`. If it is outside of this range, throw an `IllegalArgumentException`.
    2. A constructor that takes no parameters. It calls the other constructor with the following values: `AirBus`, `ROME`, `83`, and a `DateAndTime` instance initialised using the empty constructor
- Create `getFlightDuration()` which is taken from the enum field's `final` field.
- The standard textual representation of `Flight` objects has to look like this: `Flying AirBus with 83 passengers to ROME on 2021.12.22 at 11:42`

Task 2: Flyable, Plane (12 Points)

Create the interface `exam20211222.travel.flying.Flyable` with the method `estimatedArrivalTime` that takes a `Destination` and a `departHour`

, and the method `getPrice` that takes `discountRate` of type `double` and returns a `double` value.

Create the class `exam20211222.travel.flying.Plane`, a child class of `Flight` that also implements `Flyable`. It has the following fields: `name` of type `String`, `id` of type `int`, and `ticketPrice` of type `int`. All fields have getters, but not setters.

- Its single `private` constructor takes values for its fields and sets their values.
  - It throws an `IllegalArgumentException` if `name` is `null` or `ticketPrice` is less than 10.
- Write a static method `make` which takes `data` of type `String` as argument, and returns `Plane` type. `data` is in the form `name,id,ticketPrice`.
  - You do not need to check the format, you may assume that the input data is OK.
  - The method returns a `Plane` object initialized with the three components of `data`.
    - Hint: take `data` apart along the occurrences of the `,` text, then convert the resulting text sections as necessary.
- Two instances of the class are considered equal in content if the values of the three fields are matching, make sure that you override the appropriate method.
  - Also override `hashCode()` based on the values of the three fields.
- Make the textual representation of an instance very simple: `NameOfThePlane,12,83` if the `id` value is `12` and the ticket price is `83`.
- The `estimatedArrivalTime()` takes a `departHour` and a `Destination`. It returns the (integer) hour that is the estimated flight hours later.
  - Example: if the plane departs at `15` hours, and the destination is `01:45` away, the return value is `17`.
  - You may suppose that the departure and arrival happen on the same day.
- The `getPrice()` takes a `discountRate`, a `double`. It returns the `double` price which is the discounted ticket price.
  - Example: if the `ticketPrice` is `100`, and the `discountRate` is `0.2`, the return value is `80`.

Task 3: FlightWithManyPlanes (14 Points)

Create `exam20211222.travel.flying.FlightWithManyPlanes`, a child class of `Flight`. It has a field called `planes`, a list of `Plane`s.

- It has one constructor takes all the fields of the (non-zero-arg) `Flight` constructor, and an arbitrary number of `Plane`s (possibly zero, possibly a thousand of them) in an array.
  - Invoke the constructor of the base class with the arguments.
  - Fill the list with the planes with the elements of the list.
- Create `save()` which takes a `filename`. Open the file and write the following content into it.
  - The first four lines contain the textual representations of `name`, the enum value name of `destination` (hint: enums have a `name()` method), `numberOfTravellers`, and `flightDateAndTime`.
  - The remaining lines each contain the textual representation of a plane.
- Create `load()` which takes a `filename`. Open the file and read the following content from it. You may assume that the file exists, and it contains properly formatted data.
  - Read the first three lines and decode them into the appropriate fields.
  - As decoding a `DateAndTime` would be slightly more complex, you may skip the fourth line. That is, read it and ignore it.
  - Empty `planes`. Then read all remaining lines, and add them as new planes.
- Let both `load()` and `save()` throw `IOException` exception to appease the compiler.
- Create method `getCheapestRide` which takes an argument of type `double`, `discountRateIncrease`.
  - The method throws an `IllegalStateException` if the number of Planes is zero.
  - Otherwise, it finds the plane with the lowest price.
    - The first plane is not discounted at all.
    - With each further plane, the discount rate increases by `discountRateIncrease`.
    - So, the second plane is discounted at `discountRateIncrease`, the third one is at twice `discountRateIncrease` etc.
  - The method returns the reference of the cheapest plane.

This quiz was locked Dec 22, 2021 at 8pm.

Retake

Programming Languages Exam 2022-01-07 Lab

Due No due date **Points** 40 **Questions** 1
Available Jan 7 at 5pm - Jan 7 at 8pm about 3 hours **Time Limit** None

Instructions

About the exam

Remember that the conditions for the practical exams (see the file about them in the theoretical Canvas) apply.

- Specifically, don't forget to keep sending in your Code Together links using the assignment in Canvas.
  - If you would like to use Live Share: before you start sharing, in Codium, open the **File/Preferences/Settings** menu, write "connection mode" into the text input, and select the **relay** option in the dropdown.
  - Despite this, in rare cases, the share can go wrong. If you don't see the lab teacher enter the share in a few minutes, please try creating a Code Together share, and submit that using the assignment in Canvas. If even that fails (the teacher still does not appear in the share after a few more minutes), contact the teacher via Teams private chat.
- Also remember what you need to do at the end of the exam (the details of solution submission).

About the exercise

> `Pluto` is an imaginary application with many features. We will partially implement only some components that handle geographic coordinates.

Unless the assignment requires something else, proceed as "usual" during the semester:

- **the input data can be assumed to be correct, so the values do not need to be checked** (e.g. when reading from a file, in the constructors and methods etc.);
- the visibility of the fields is as narrow as possible;
- the visibility of the methods, constructors and types is as wide as possible;
- "integer" means `int`, "real" means `double`;
- make sure you implement encapsulation correctly to avoid data leakage.

Depending on your environment, real numbers may or may not have decimal points (i.e., `3.1415`) when

Depending on your environment, real numbers may or may not have decimal points (i.e., `3.1415`) when written into strings, on screen, or to text files, but decimal comma (i.e., `3,1415`). This, however, should not cause a problem and does not affect your results evaluation.

Testing

Testing is also part of the assignment. The required files can be downloaded here.

- The JUnit 4 test codes are given. Test codes should not be modified except for the following:
  - The parts marked with `// TODO` must be implemented/completed, as part of your tasks.
  - Test cases for which the implementation has not yet been completed may be commented out temporarily.
  - Most test cases are originally commented out. These should be uncommented gradually as you proceed.
- Use test cases to check your solution. Please note that not all possible scenarios are covered with test cases.
- Here's how to run test cases on Windows if, for example, our testing class is `pluto.geo.primitive.PlutoTester` (on Linux, use `:` instead of `;`):

```
javac -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." pluto\geo\primitive\PlutoTester.java
java -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." org.junit.runner.JUnitCore pluto.geo.primitive.PlutoTester
```

1. Degree, Direction, GeoException (12 points)

Degree

Create the type `pluto.geo.primitive.Degree`.

> We model plane angles (see e.g. https://en.wikipedia.org/wiki/Angle#Individual_angles) with this type.

> In this type, the magnitude (i.e., value) of the angle is stored both as a real number ***and*** in the form of degrees-minutes-seconds ("deg-min-sec" or "dms"). For example, when modeling degrees `90.5` (ninety and a half), we store the real number `90.5` on the one hand, and the integer `90` (degrees, deg), integer `30` (minutes, min) and integer `0` (seconds, sec).

- Visibility of this type should be the default visibility.
- Fields: real `degree` and integer `deg`, `min`, `sec`. Fields should also have the default visibility. `degree` has a getter, but not setter. Other fields have neither getter nor setter.

getter, but not setter. Other fields have neither getter nor setter.

- The type must have a constructor with a real parameter that checks the value of this parameter (see below) and then assigns the value to the appropriate field.
  - The constants `MIN_VALUE` and `MAX_VALUE` of type `Integer` designate a subset of integers. If the parameter value is not element of this subset, the constructor shall throw an exception of type `IllegalArgumentException` with the message `"value X is out of range MIN..MAX"` (where `MIN` and `MAX` are the values of the constants mentioned and `X` is the value under test). No other check is required.
- The constructor with no parameters calls the other constructor with value `0.0`.
- Create the private, no-parameter method `setDegMinSec` without return values. The purpose of this method is to assign a value to the fields `deg`, `min` and `sec` based on the value of `degree`:
  1. first calculate the value of degrees and then store it in the `deg` field;
  2. then calculate the value of minutes and store it in the `min` field;
  3. then calculate the value of seconds and store it in the `sec` field;
- The one-parameter constructor calls the `setDegMinSec` method at the appropriate location.
- Make textual representation of this type in the form of `"90d 30m 0s"` (this example shows `90` degrees, `30` minutes, `0` seconds).

> Help to calculate angular minutes and angular seconds from a real angle value (e.g. `90.5` degrees): https://www.rapidtables.com/convert/number/degrees-to-degrees-minutes-seconds.html

Direction

Create enumeration type `pluto.geo.Direction` to handle equatorial regions:

- Elements of this type: `NONE`, `NORTH`, `EAST`, `SOUTH`, `WEST` (in this order).

> `NONE` is the extremal element (meaning ca. "No direction is set").

GeoException

Create type `pluto.geo.GeoException` that represents our own unchecked exception:

- The type must be a subtype of `IllegalArgumentException`.
- The type should have two constructors: one with no parameters and one accepting a `String` parameter.
  - The latter constructor calls the appropriate constructor of the parent type.

2. GeoDegree, Latitude, Longitude, Place (13 points)

> Type `Degree` was created for modeling plane angles. We are now creating additional classes: type `GeoDegree` (base) for modeling geographic degrees and types `Latitude` and `Longitude` for modeling, well, geographical latitudes and longitudes.

GeoDegree

Create abstract class `pluto.geo.primitive.GeoDegree`.

- Fields: `degree` (`Degree`) and `direction` (`Direction`). Visibility of both these fields should be as narrow as possible while they should still be available from the child classes.
- Let the class have class constants `MAX` and `MIN` of type real. `MAX` should be `+ 360.0`, `MIN` should be `MAX` negated. Their visibility is the same as the fields above.
- The only constructor expects a `Degree` and a `Direction` in this order. It stores the values.
- Create getters for instance fields. The getter for `degree` returns a real number.
- Class constants should also have queries: create class methods `getMin` and `getMax`.
- The textual representation of the class is: concatenating the name of the value in the `direction` field, then a space, then the textual representation of the `degree` field (e.g. `"WEST 3d 14m 15s"`).
- The class should have an abstract, no-parameter method `getVisual` with a parameter `String`.

Latitude, Longitude

Create types `Latitude` and `Longitude` in package `pluto.geo.primitive` as subtype of `GeoDegree`. (These are to model latitude and longitude values.)

Visual help for navigation:

```
Latitude:

   +90
    ^
    |
  NORTH(+)
    |
    0
    |
  SOUTH(-)
    |
    v
   -90
```

```
Longitude:

    -180 <-- WEST(-) -- 0 -- EAST(+) --> +180
```

> These two classes are very similar.

- Both classes should have a public constructor that expects a real `degree` as a parameter. This constructor calls the constructor of the parent class:
  - To do this, a `Degree` object must be instantiated when calling the superclass's constructor and the `Direction` must also be derived based on the sign of `degree` (see also the table below).
  - For `Latitude` objects, a negative `degree` parameter means `SOUTH`, a non-negative one means `NORTH`.
  - For objects of type `Longitude`, a negative `degree` parameter means `WEST`, a non-negative means `EAST`.
- The textual representation of this types is made by appending the `degree` field as a real number after `"LAT="` (for `Latitude`) and `"LON="` (for `Longitude`). (Therefore, the textual representation of the parent type is not used in this method.)
- Abstract method `getVisual` is implemented in the same way in both classes: it returns with the textual representation of the parent type.

class **sign of value direction**

class	sign of value	direction
`Latitude`	`0`, `+`	`NORTH`
`Latitude`	`-`	`SOUTH`
`Longitude`	`0`, `+`	`EAST`
`Longitude`	`-`	`WEST`

Place

Implement class `pluto.geo.Place` for modeling well-known geographic locations (e.g. cities, statues, schools, etc.):

- A place has a name (in string field `name`) and two `GeoDegree` fields for storing latitude and longitude coordinates (`latitude`, `longitude`). All three fields have getter, but no setter.
- The constructor accordingly expects a string, a value of type `Latitude` and a value of type `Longitude` in this order and stores them in the appropriate fields.
- Implement textual representation of the type in the following format:

```
Budapest:LAT=47.498333:LON=19.040833
```

3. PlaceRegister (15 points)

Implement class `pluto.geo.PlaceRegister`.

- This type has a class variable with type list called `places`. We can store places (`Place`) in this data structure. Implement a getter for this field.
- Implement public class method `loadFromFile`, which processes a text file line by line whose named is provided as parameter (`filename`, `String`):
  - Make sure you manage your resources properly.
  - The method can throw `IOException`s that may occur.
  - Lines starting with `#` are comments in the file. These lines do not need to be processed.
  - It is assumed that each non-comment line contains the textual representation of a `Place`.
  - This method overwrites any existing contents of `places`.

> It is a good idea to read an entire line first and then decide whether to process it. If you need to process the content of the line, you may want to break it down into smaller parts (see separator characters). It is recommended to use the appropriate method of the `String` class for this (see *Java API* documentation).

> (**Optional**) You can improve the clarity of your code by creating a separate private method to process a line that has been read from the file.

Example file:

```
Budapest:LAT=47.498333:LON=19.040833
New York:LAT=40.716667:LON=-74
Tokio:LAT=35.689444:LON=139.691667
# We have 1 more place: Rio!
Rio de Janeiro:LAT=-22.908333:LON=-43.196389
```

- Implement the public, class method `saveToFile`, which prints the contents of the `places` data structure to the text file in its parameter (`filename`, `String`).
  - Each line of the output file is a textual representation of a `Place` object.
  - The method can throw `IOException`s that may occur.
  - Make sure you manage your resources properly.

This quiz was locked Jan 7 at 8pm.

Retake 2

Due No due date **Points** 40 **Questions** 1
Available Jan 20 at 5pm - Jan 20 at 8pm about 3 hours **Time Limit** None
Allowed Attempts Unlimited

Instructions

Programming languages Java exam, 2022-01-20

About the exam

Remember that the conditions for the practical exams (see the file about them in the theoretical Canvas) apply.

- Specifically, don't forget to keep sending in your Code Together links using the assignment in Canvas.
- Also remember what you need to do at the end of the exam (the details of solution submission).

About the exercise

Where the assignment does not tell you otherwise, then create your code to be as good as possible, following the practices we used in the semester:

- **we can assume, that data is correct, so you do not have to check it (for example: reading from file, constructor, in the methods, etc.)**
- the visibility of the fields is the narrowest possible
- the methods, constructors and the types visibility is the widest possible
- "integer" means `int`, "real" means `double`
- look out for the correct encapsulation and avoid leaking data

Testing

The testing is also a part of the assignment. You can download the necessary files from here.

- We give you the JUnit 4 testing codes in this exam. You may not change the test codes.
- In the beginning, most of the tests are commented out. As you are solving the exam, you have to delete the comments.
- Use the test cases to check your solution. Note that tests cannot fully guarantee that your code is good.

- Run the test like this:
  - On Windows:

```
javac -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." preri\test\PreriTester.java
java -cp "junit-4.12.jar;hamcrest-core-1.3.jar;." org.junit.runner.JUnitCore preri.test.PreriTester
```

  - On Linux:

```
javac -cp "junit-4.12.jar:hamcrest-core-1.3.jar:." preri\test\PreriTester.java
java -cp "junit-4.12.jar:hamcrest-core-1.3.jar:." org.junit.runner.JUnitCore preri.test.PreriTester
```

1. Ingredient, PizzaSize, Food (10 points)

Ingredient

Make the `preri.pizza.utils.Ingredient` class that represents a food's ingredients.

- The fields of the class:
  - `price` integer
  - `name` and `amountName` (the quantitative unit of the ingredient) string
  - `amount` real (the amount of ingredients)
  - The visibility of the fields must be the default and they must have `getter`s and `setter`s
- The class must have a constructor which gets the contents of all of the fields and makes these inspections:
  - the strings can only contain letters
  - the `amountName` field **can not** be an empty string
  - the `name` field must be at least `3` letter long
  - the `amount` and `price` fields value must be bigger than `0`.
  - If some of the conditions are not fulfilled, then the constructor throws an `IllegalArgumentException` with the message `Invalid argument!`

PizzaSize

Make the `preri.pizza.utils.PizzaSize` enumeration type for the size of the pizzas:

- The elements of the type are the following : `SMALL`, `MEDIUM`, `LARGE`.

Food

Make the `preri.pizza.Food` interface which contains the following methods:

- `getPrice` returns with an integer value
- `getName` returns with a string value

2. Pizza, Cart, CartException (14 points)

Pizza

Make the `preri.pizza.Pizza` class which contains the following fields:

- `size (PizzaSize type)` which stores the size of the pizza
- `name` is string
- the ingredients are in a `Set<Ingredient>`, the name of the field is `ingredients`

It must have two constructors, one of it just waits for a size from the type of `PizzaSize` and `"Pizza"` is the default name. The other constructor waits for the name of the pizza `(String)`, the size of the pizza `(PizzaSize)` and the ingredients `(Set<Ingredient>)`, after that it assigns the value of the fields.

The class implements the `Food` interface. The methods of the interface must be like this:

- `getName`: must return with a pizza name
- `getPrice`: `cost = 3*sum(ingredients)+size*size`

Also, the class must have a `getIngredients` method which returns with the ingredients of the pizza in a `Set<Ingredient>` data structure.

CartException

Make the `preri.pizza.utils.CartException`, a user defined exception class:

- The class extends from the `Exception` class.
- The class must have two constructors: one without parameter, and one which waits for a `String`.
  - The last one must call a suitable constructor of the parent class

Cart

Make the `preri.pizza.Cart` class which has a `cart` field and its type is `Map<Pizza, Integer>`. The class implements the following methods:

- `add`: it waits for an object with the type of `Pizza`, if the cart does not contain it, then put in the cart with multiplicity of `1`, if the cart already contains a pizza like this, then then it increases the multiplicity.
- `remove`: it waits for an object with the type of `Pizza`, if the cart contains only `1` of it, then removes it from the cart, otherwise decreases the multiplicity with `1`. If it gets a pizza as a parameter, that is not in the cart, then the methods throw an `CartException` with a message like this: `"Cannot remove this item: " + PIZZA_NAME + ", because it is not present in the cart."`, where the `PIZZA_NAME` is the name of the pizza that it gets from the parameter. The method throws the `CartException`.
- `getCount`: it waits for an object with the type of `Pizza`, then returns with the multiplicity of the pizza.
- `getTotalCost`: returns with total cost of the cart.

3. equals, hashCode, toString (11 points)

Ingredient

Implement the equality comparison method for `preri.pizza.utils.Ingredient`. Two copies of the class are equal to each other if they have the same `price` and `name`.

- Also implement the `hashCode()` method based on the value of the fields.

Pizza

In a similar fashion, implement the equality comparison method `preri.pizza.utils.Ingredient` based on its three fields.

- As above, implement `hashCode()`, too.

Cart

Override the `toString` method of the `Cart` class like this:

```
The contents of the cart:
  Margarita (2x): 2000Ft
  Songoku (1x): 1000Ft

The total price of the cart is: 3000Ft
```

Make sure that you put in the line breaks (`\n`) and the indentation (the pizzas start two spaces in).

4. loadFromFile (5 points)

Make a static method with the name `loadFromFile` in the `Pizza` class which is able to read from a text file like this:

> `PIZZA_NAME`:`PIZZA_SIZE`:`INGREDIENT_NAME`-`PRICE`-`AMOUNT`-`AMOUNT_NAME`

and it returns with `List<Pizza>`. The method takes one parameter, the path of the file (`String`). You may assume that the file exists. Make sure that your code doesn't waste resources. The data is like this in every line:

The pizza data is separated with `;`, the data of an ingredient is separated with `-`. You may assume that the data in the file is correct.

```
Hawaii;SMALL;Dough-150-500-g;Tomato-100-200-g;Pineapple-300-15-dkg;Cheese-150-10-dkg
Songoku;MEDIUM;Dough-250-75-dkg;Tomato-150-250-g;Ham-300-150-g;Corn-100-10-dkg;Mushroom-400-0.2-kg;Cheese-200-18-dkg
Margherita;LARGE;Dough-350-1-kg;Tomato-250-350-g;Cheese-300-25-dkg
```

This quiz was locked Jan 20 at 8pm.