1. Homework 5

Posted: September/23/2018
Due: September/30/2018 24.00

All homework solutions are due September/30/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Homework 5.1 (10 Points)

Objective: Learn ho to modify an existing hierarchie

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to **100**% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Note the change in point deduction.

In hw 4.1 you designed a system using inheritance, and/or abstract classes, and/or interfaces. Most requirements of a system change over time. You have to modify your solution from hw 4.1 to me the new requirements.

Explanation:

You have to add to the existing functionality the following animals:

- a *Polar Bear* which is a *Carnivore*,
- a *Ferret* which is a *Carnivore*,
- a *Alpaca* which is a *Herbivore*,
- a *Camel* which is a *Herbivore*,
- a *Skunk* which is a *Omnivore*,
- an *Aardvark* which is a *Omnivore*,

A Omnivore is a Carnivore and a Herbivore. Meaning a omnivore can hunt and graze.

It is recommended that you draw the class diagrams before you start to code.

Your Work:

It is assumed that your solution of hw 4.1 can not easlily be modified to fit the new criteria. You have to rethink your design for hw 5.1

Requirements:
You have to name your main class Zoo.java and you have to name your classes in a reasonable way. You have to be able to explain your class diagram to the grader.

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab5-1 'All files required'
# you can see if your submission was successful:
# try -q hpb-grd lab5-1
```

1.2. Homework 5.2 (10 Points)

Objective: Learn ho to modify an existing hierarchie

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to **100**% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Note the change in point deduction.

In hw 5.1 and 4.2 you designed a system using inheritance, and/or abstract classes, and/or interfaces. Most requirements of a system change over time. You have to modify your solution from hw 5.1 or 4.2 to meet the new requirements.

Explanation:

A new zoo director is hired and the new director decided that all animals and plants need to have a serial id. A serial id is a unique number which identifies all animals and plants.

- A serial id for a school of fish is defined as: school id and how many fish are in this school.

The director wants to be able to:

- to send the animals out to eat or home.
- ask the animals where they are. Animal and plants are at x/y positions in the Zoo. X/y can describe a large area.
- Move the animals and moveable plants to new positions.

The zoo has:

- a *Polar Bear* which is a *Carnivore*,
- a *Alpaca* which is a *Herbivore*,
- an *Aardvark* which is a *Omnivore*,
- one school of gold fish,
- one palm tree in a pot.

A Omnivore is a Carnivore and a Herbivore.

Design the system so such the zoo director can do what they director wants.

You will notice during the design that not everything you want to be specified is specified. You can make your own decision, but you have to

- document them
- be able to defend them during you grading session.

It is recommended that you draw the class diagrams before you start to code.

Your Work:

Requirements:
You have to name your main class Zoo.java and you have to name your classes in a reasonable way. You have to be able to explain your class diagram to the grader.

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab5-2 'All files required'
# you can see if your submission was successful:
```

```
# try -q hpb-grd lab5-2
```

1.3. Homework 5.3 (10 Points)

Objective: Designing a Storage System Using Generics

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to **100**% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

Note the change in point deduction.

You have to implement a storage system which allows to storage and remove items using generics.

In particular you have implement the following methods of a storage system:

```
boolean add(E e)
void    add(int index, E element)
void    addFirst(E e)
void    addLast(E e)
void    clear()
E       element()
E       remove()
E       remove(int index)
int     size()
```

A documention of the methods can be found

As you noticed a constructor is not defined. It is up to you to define at least one constructor.

You can not use any excisting Java class for this.

Explanation:

The system you are asked to design and implement does not have a fixed data size. A linked list might be a useful data structure.

Your Work:

You need to develop the Storage class and a test framework. A potential framework might look like:

```
    public static boolean testAdd()     {
        Storage<String> aStorage = new Storage<String>();
        String theStrings[] = { "a", "b", "c" };
        boolean rValue = true;
        for ( int index = 0; index < theStrings.length; index ++ )
                aStorage.add(theStrings[index]);
        for ( int index = 0; index < theStrings.length; index ++ )
                rValue &= aStorage.remove().equals(theStrings[index]);
        rValue &= aStorage.remove() == null;
        aStorage.add("c");

        return rValue;
    }
   public static void exampleOfHowToUseIt( Storage<String> aStorage)   {
        aStorage.add("a");
        aStorage.add(0, "0");
        aStorage.add(aStorage.size(), "1");
        aStorage.add(aStorage.size() + 1, "2");
        System.out.println("aStorage: " + aStorage );
```

```
    }
    public static void test(Storage<String> aStorage)   {
                if ( ! testAdd() )
                        System.err.println("testAdd failed");
                if ( ! testRemoveIndex() )
                        System.err.println("testRemoveIndex failed");
                if ( ! testAddIndex() )
                        System.err.println("testAddIndex failed");
                if ( ! testClear() )
                        System.err.println("testClear failed");
    }
    public static void main(String args[] )     {
        test(new Storage<String>());
        exampleOfHowToUseIt(new Storage<String>());

    }
```

Requirements:

You need to design and implement O(1) algorithms when ever possible. It might be difficult for methods the methods *remove(int index)* and *add(int index)* to implement a O(1) algorithm. You have to name the Storage class *Storage.java*.

Example:

An example of a solution execution:

```
% java Storage
aStorage: # of elements: 3  ->0->a->1->null
%
```

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab5-3 'All files required'
# you can see if your submission was successful:
# try -q hpb-grd lab5-3
```