Kursusgang 2 - Nedarvning

1 Access Modifiers

(a) Explain, in your own words, the access modifiers: private, protected, and public. When would you use each? Which would you use by default?

2 Salary Calculator

Write a class to represent an employee. An employee has a name, job title, and salary. Write a subclass to represent a manager. A manager has a yearly bonus.

(a) Add appropriate constructors for the classes. Discuss in the group what appropriate means

(b) Employee has a method **CalculateYearlySalary;** implement this

(c) For a manager, **CalculateYearlySalary** includes the yearly bonus

(d) Employees and managers have seniority levels 1-10. Each level results in 10% extra salary, e.g. level 3 is 30% and level 7 is 70% extra. Bonus is not affected by seniority level.

3 List Filtering

In this exercise we will design a list-filtering framework for filtering lists of person, List<Person>. The filtering is used like this:

```
1  //With a Person-list:
2      List<Person> plist = new List<Person>();
3      plist.Add(new Person("Thomas",...));
4      plist.Add(new Person("Erik",...));
5      plist.Add(new Employee("Bo",...));
6      plist.Add(new Employee("Hans",...));
7      plist.Add(new Person("Kurt",...));
8  //I can retrieve a list of all named ''Thomas'' using:
9      PersonFilter pfilter = new NameFilter("Thomas");
10     List<Person> filteredList = pfilter.Filter(plist);
```

Listing 1: Filtering usage

(a) Create an abstract class PersonFilter

(I) Add an abstract method **bool FilterPredicate(Person person)**

(II) Add a virtual method **List<Person> Filter(List<Person> plist)**. This method should provide a default filtering-implementation, where the FilterPredicate-method above is used to decide if a person belongs to the filtered result. Spend time designing this method. If you are stuck, there is a suggestion on the last page.

(b) Create the following filter-implementations:

(I) The NameFilter, as used on Listing 1

(II) An age filter, parameterized with Min and Max-age

(III) EmployeeFilter; selects all employees

(IV) Not-filter

i. Parameterized with a filter X, it does the opposite of X, e.g. parameterized with the filter in the example above, it will return all people **NOT named "Thomas"**

(V) And-filter

i. Parameterized with two filters, only if both filters agree, the element is part of the result. E.g. parameterized with Filter X and Not(Filter X) will always produce an empty list.

(VI) Or-filter Same as above, but instead FilterPredicate is as follows:

```
public override bool FilterPredicate(Person person)
        => pf1.FilterPredicate(person) || pf2.FilterPredicate(person);
```

Listing 2: Or-filter

(VII) Xor-filters Same as above, but instead FilterPredicate is as follows:

```
public override bool FilterPredicate(Person person)
        => pf1.FilterPredicate(person) ^ pf2.FilterPredicate(person);
```

Listing 3: Xor-filter

(c) Pass-through filter: I have added the following sub-type, used for printing.

(I) Modify the above class and ensure all subtypes of PassThroughFilter returns the list unmodified

```
public virtual List<Person> Filter(List<Person> people)
    {
        List<Person> result = new List<Person>();
        foreach (Person person in people)
        {
            if (FilterPredicate(person))
                result.Add(person);
        }
        return result;
    }
```

Listing 4: Filter Suggestion

4 Parking Meter

Write a class to represent a parking meter. The parking meter should have a method to insert coins and pay for x minutes. The parking rate depends on whether it is weekday or weekend.

(a) Write an abstract class to capture the computation of the parking rate. Use the abstract class in the parking meter to calculate rate.

(b) Write two classes, which extend the abstract class, one for the rate in weekdays and one for the rate in weekends.

5 Bank Account

Write a class to represent a bank account.

(a) A bank account has a balance, a borrowing rate, and a savings rate.
The borrowing rate might be 10% but the savings rate might be only 1%.

(b) Add methods to deposit and withdraw money.

(c) Add a method to accrue or charge interest depending on the current balance.

(d) Ensure, via proper encapsulation, that the following invariants are true:

(I) the balance must never be less than -100,000,

(II) the balance must never exceed 250,000,

(III) you cannot deposit or withdraw a negative amount of money

(IV) the borrowing rate must be at least 6%. The savings rate must be at most 2%