Inheritance exercises

1. Explain, in your own words, the access modifiers: private, protected, and public. When would you use each? Which would you use by default?
2. Write a class to represent an employee. An employee has a **name, job title** and a **salary**. Write a subclass to represent a manager. A manager has a yearly bonus **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 manager, **CalculateYearlySalary** includes bonus
   d. Employe-and managers has seneniority levels 1-10, each level results in 10% extra salary. Level 3 is 30%, level 7 is 70% extra. Bonus is not affected by seniority levels.
3. 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. Write an abstract class to capture the computation of the parking rate. Use the abstract class in the parking meter to calculate rate. Write two classes, which extend the abstract class, one for the rate in weekdays and one for the rate in weekends
4. 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 be more than 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%
5. In this exercise we will design a list-filtering framework for filtering lists of person, List<Person>. The filtering is used like this:

```
With a Person-list:
            List<Person> plist = new List<Person>();
            plist.Add(new Person("Thomas",...));
            plist.Add(new Person("Erik",...));
            plist.Add(new Employee("Bo",...));
            plist.Add(new Employee("Hans",...));
            plist.Add(new Person("Kurt",...));
```

I can retrieve a list of all named “Thomas” using:

```
            PersonFilter pfilter = new NameFilter("Thomas");
            List<Person> filteredList = pfilter.Filter(plist);
```

   a. Create an abstract class PersonFilter, with an abstract method
      bool FilterPredicate(Person person)

b. 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.

c. Create the following filter-implementations:

   i. The NameFilter, from the above example

   ii. An age filter, parameterized with Min and Max-age

   iii. EmployeeFilter, select all employees

   iv. Not-filter

      1. 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

      1. 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

   vii. Xor-filters

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

```
abstract class PassThroughFilter : PersonFilter
{
    public override List<Person> Filter(List<Person> plist)
    {
         foreach (Person person in plist)
             FilterPredicate(person);
        return people;
    }
}
```

   i. Modify the above class and ensure all subtypes of PassThroughFilter returns the list unmodified

Hint:

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

    public abstract bool FilterPredicate(Person person);
}
```