Home-work 6

1. **This question was asked in a previous exam!**
   Assume we have an operator `\type(x)`, which returns the dynamic type of the reference `x`. The operator `<:` expresses the subtype-relation. `S <: T` holds, if `S=T` or `S` is a subtype of `T`. You can use these operators to express the typing of a program in the interface-definition with preconditions, postconditions, and invariants.

```
class Exa {
   Object a;

   Object m(Object c) { ... }
}

class ExaTyped {
   A a;

   B m(C c) { ... }
}
```

   1. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `C` (with subtypes) can be used as parameter for `m.` Do not change the typing of `Exa`.

   2. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `B` (with subtypes) will be given back as return value for `m.` Do not change the typing of `Exa`.

   3. Specify the class Exa with the help of the operators `\type(x)` and `<:` in a way, such that only objects of type `A` (with subtypes) can be stored in the field `a.` Do not change the typing of `Exa`.

   4. Is the specified version of the class `Exa` (the solution of the previous question) equivalent to the stronger typed class `ExaTyped`? Give an implementation of the method `m` showing the differences.

2. **Co-, Contra-, and In-variant example from a previous exam.**
   Given is the following program-fragment of a Java-like programming language:

```
class Super {
   B m(C c) { ... }
}
class Sub extends Super {
   E m(F c) { ... }
}
```

This programming language allows you to adopt the parameter and return type of methods in subclasses. The program language designer can not determine which rules he has to enforce to guarantee static type safety.

a. Which relationship has to be between the parameter types `C` and `F`? Is this rule co-, contra- or in-variant?

b. Which relationship has to be between the return types `B` and `E`? Is this rule co-, contra- or in-variant?

c. For each of them give a code-fragment showing the problems, which arise without such restrictions.