INF3110

Exercises OO-II, 2016

Problem 1

Sketch a solution to 10.1 a), b) and c) in Mitchell book.

Problem 2

Consider these two classes:

```
class Rect {
  Point ul;       // upper left corner
  Point lr;       // lower right corner

  void setUL(Point newUL){ this.ul = newUL;};
};

class ColorRect extends Rect {
  ColorPoint ul;
  ColorPoint lr;
}
```

a) In Java this is allowed, however, it does not imply that the types of the inherited 'ul' and 'lr' are redefined to become ColorPoint instead of Point. Instead a ColorRect object will have two variables with name 'ul' (with types Point and ColorPoint), and two variables with name 'lr' (with types Point and ColorPoint).

Supposed that Java were changed so that the types of the inherited 'ul' and 'lr' would be redefined to ColorPoint. What kind of type error would that imply?

b) How would this be done with virtual classes?

A virtual class is a class (in a superclass) that can be redefined in the same way as methods can be redefined in subclasses, with the constraint that it can only be redefined to subclasses of the constraint the original class. In the example with virtual classes from the lecture (Point/Colorpoint, slide 35 of OO-I) this implies that the virtual class Type can only be redefined to subclasses of Point, which is the constraint of Type.

c) Virtual classes are not part of Java. Would a cast help, like in this redefinition of setUL in ColorRect:

```
class ColorRect extends Rect {
  void setUL(Point newUL){this.ul = (ColorPoint)newUL;};
}
```

Problem 3

a) Is there any alternative to multiple inheritance if only re-use of code from (super) classes is wanted, not subtyping?

Use the Stack, Queue and Dequeue example and try to base Dequeue on Stack and Queue.

Use the same technique to make both Stack and Queue based upon Dequeue.

b) What if one would also like to have the subtyping relationship, so that references typed with e.g. Stack and Queue can denote objects of the resulting class?

Problem 4

A Java array of type T is declared by T[]. The Java subtype rule for array types is

S'[] subtype of S[] if S' subtype of S

So, Java arrays are covariant, in contrast to generic classes.

Suppose we have class C with subclass CSub and that the method 'methodOfCSubOnly()' is defined in CSub only and not in C.

Look at

```
class TypeTest {
  C v = new C();
  void arrayProb(C[] anArray) {
    if (anArray.length >0)
      anArray[0] = v;                    // (2)
  };
  static void main(string[] args) {
    TypeTest tt = new TypeTest();
    CSub[] paramArray = new CSub[10];
    tt.arrayProb(paramArray);            // (1)
    paramArray[0].methodOfCSubOnly();    // (3)
  };
}
```

What happens? Especially at lines (1) (2) and (3)?

If Java had insisted on complete static type checking in this case, what would then be the requirement on the actual parameter to the call 'tt.arrayProb(paramArray)'.

Problem 5

Suppose that we have class Reservation with subclasses FlightReservation and TrainReservation as described in the foil set. As part of a reservation system it is desirable to have a collection of reservations that cater for possibly new subclasses for new kinds of reservations (e.g. for space travels).

How would you make a print method that prints all elements of such a collection, using the generics mechanisms of Java?

Problem 6

We have seen structural type compatibility and subtyping wrt e.g. Smalltalk, as described in Mitchell. Objects of classes have the same interface in terms operations if they provide the same set (or subset) of operations.

Consider the following Java sketch:

```
interface cowboy {void draw(); ...}
interface shape {void draw(); ...}

class LuckyLuke implements cowboy, shape {...}
```

Is this an example of structural (sub)typing, given the fact that Java may very well get the same method from different interfaces, but still only provide one implementation?

Problem 7

The FlightReservation class we have seen a couple of times has a Flight attribute. We assume that this is a reference to an object of class Flight. The Flight object represents the actual flight reserved.

In the flight table of SAS we have entries for e.g. SK451 (Oslo to Copenhagen). Suppose that we would like to represent such an entry by means of a FlightType object. Class FlightType would therefore have attributes that are common to all SK451 Flights, like source, destination, scheduled departure time (8.20), scheduled flying time (1.10), scheduled arrival time, etc.

SK451 takes place every day (or almost), so a reservation system would need to have one Flight object for each actual flight. These Flight objects will have a representation of seats (free, occupied), and for other reasons one may imagine that they will also have actual departure time, actual flight time and delay (departure and arrival delay).

It is perfectly possible to do this without inner classes, but if you should exploit inner classes, how would this be done. Of special interest are of course the functions computing the departure and arrival delays.

Feel free to be inspired by the slide on inner classes exemplified by class Apartment, specially the fact the attribute hight of Apartment is visible in the inner classes. Attributes like scheduled departureTime and arrivalTime should be attributes of the outer class FlightType.