PLD Assignment 3

Ask

18. marts 2022

Indhold

1 A3.1

1.1 a)

The compiler rejects the program with a compile error, since the method "bingoString()"isn't defined for the generic type T.

This is the error message I get when I try to compile the program

```
Bingo.java:9: error: cannot find symbol
    System.out.println(t.bingoString());
                       ^
symbol: method bingoString()
location: variable t of type T
where T is a type-variable:
T extends Object declared in class Bingo
1 error
```

1.2 b)

The following code should compile. But throw a NullPointerException. Otherwise, since we don't really know what BingoString is supposed to do in this specific code snippet, a runtime error could be cause by a wrong cast.

```
1  abstract class myAbstractClass {
2      public abstract String bingoString();
3  }
4
5  class Bingo<T extends myAbstractClass> {
6
7      public void dingo(T t) {
8          System.out.println(t.bingoString());
9      }
10
11
12     public static void main(String[] args) {
13         Bingo<myAbstractClass> myObj = new Bingo<>();
14         myObj.dingo(null);
15     }
16 }
```

1.3 c)

Reusing the same piece of code, as in 1b only with a minor tweaks. This code will compile but do nothing.

```java
class myClass {
    public String bingoString(){
        return "This is bingo string, who dis?";
    }

}

class Bingo<T extends myClass> {

    public void dingo(T t) {
        System.out.println(t.bingoString());
    }

    public static void main(String[] args) {
        Bingo newBingo = new Bingo();
        newBingo.dingo(new myClass());
    }
}
```

1.4 d)

Since we assume that the code will compile, this should lead to a run-time error. since we can't set a subclass equal to the value of a superclass in java.

1.5 e)

2 A3.2

2.1 a)

Advantages

- Extensive control over the program and which parts of the program that should be able to use which modules. Can change the export statement in descendant classes.
- Security. Having the prosibility to restrict certain parts of the program from using specific moduels. (This looks very similar to public, private, protected as we see in Java. Only at a slighy more general level in Eiffel.)

Disadvantages

- Risk of making to many internal functions unusable to other parts of the program. And or making the readability of the different exports a hassle to understand.
-

2.2 b)

Advantages

- Since every program in Modula-2 is composed of moduels, code can easily be resused.
- Encapsulation. This gives the programmer the option to restrict visibility of cartain parts of the code e.g. subprograms or data structures to other parts of a program.
- The possibility to only import specific methods from a module.

Disadvantages

- Importing a whole module only to use a couple of methods, may take up file size.
- Assuming that modula-2 does not import all methods by default. The programmer would need to specify all modules in case a module contains many functions.
- The programmer must keep track of both input and output of the specific module.

3 A3.3

3.1 a)

Plop is a function, that takes an empty list and some value and insert that value into the list. In the "second"line plop is called with a list pattern and the argument w. And then insert U and V in the start of the list and then insert w at the end of the instantiated list.

3.2 b)

4 A3.4

4.1 a)

Root and mixed are two different functions. Root creates a the that has a root at X with either zero, two or three children. Mixed is a function that checks whether the tree created by the root function is a mixed tree. And will possibly return a boolean value if the tree created by root is or is not indeed a mixed tree.

4.2 b)

```
root("gorilla",
    root("goat",
        root("duck",
            root("koala"), Root("manatee")),
        root("impala")
    )
    root("horse"), Root("ostrich"))
```

4.3 c)

Creating a method called: leftmost and implementing this in prolog:
I have the following.

```
leftmost(root(E),E).
leftmost(root(_, T1, _), E) :- leftmost(T1, E).
leftmost(root(_, T1, _, _), E) :- leftmost(T1, E).

mixed(root(X, T1, T2, T3)) :- string(X), mixed(T1), mixed(T2),
 ↪  mixed(T3).
mixed(root(X, T1, T2)) :- string(X), mixed(T1), mixed(T2).
mixed(root(X) :- string(X).
```

Query for calling the prolog function

```
leftmost(root("gorilla", root("goat", root("duck",
 ↪  root("koala"), root("manatee)), root("impala")),
 ↪  root("horse"), root("ostrch)), "koala")
```

4.4 d)

5 A3.5

6 A3.6

Older programming languages like Fortran-77 had the possibility for dynamic scoping with static type checking.

One possible solution would be to have variables declared in a specific manner. This could be done by having the different types only use certain variables.

An example of this could be. Integers taking the variables of: i, j, k , l, m, n Doubles taking the variables of: d, h, v, x

and so on. Note that these variables given above are just an example.