Exempel på examinationsuppgifter 1dv607

// Construct a correct UML class and sequence diagram given the following pseudo code

```
Candy treat = new Candy()
treat.doBar()

class Foo {
   void doCandy(Fuu candy) {
      candy.bar()
   }
}

class Candy : Foo {
  Fuu candy;

  void doBar() {
    candy = new Fuu();
    doCandy(candy)
  }
}

class Fuu {

  bool bar() {

    return true/false;
  }
}
```

// Construct a correct UML class and sequence diagram given the following pseudo code

```
Candy treat = new Candy()
treat.doBar()

class Foo {
   Fuu getCandy() {
      return new Fuu()
   }
}

class Candy : Foo {
  Fuu candy;

  void doBar() {
    candy = getCandy()
    candy.bar()
  }
}

class Fuu {

  bool bar() {

    return true/false;
  }
}
```

// Construct a correct UML class and sequence diagram given the following pseudo code

```
Candy treat = new Candy()
treat.doBar()

class Foo {
   void doCandy(Fuu candy) {
      candy.bar()
   }
}

class Candy : Foo {
  Candy bar;

  void doBar() {
    Candy bar = new Candy()
    bar.doTreat()
    doTreat()
  }

  void doTreat() {
    Fuu candy = new Fuu()
    doCandy(candy)
  }
}

class Fuu {

  bool bar() {

    return true/false;
  }
}
```

// Construct a correct UML class and sequence diagram given the following pseudo code

```
Fuu candy = new Fuu()
candy.bar()

class Foo {
   bool doCandy() {
       return true/false
   }
}

class Candy : Foo {
  Fuu candy;

  void doBar(Fuu bar) {
    candy = bar;
    doCandy()
  }
}

class Fuu {

  bool bar() {
    Candy c = new Candy()
    c.doBar(this)
  }
}
```

```
//
// This code is functional but has a glaring problem
// Your task is to identify the problem and refactor it using
// a standard OO-method. You should update the code and make a
// correct UML class diagram over your solution
//

// example of usage
Fuubar fb1 = new Fuu()
Fuubar fb2 = new Bar()
...
fb1.DoStuff()
fb2.DoStuff()

interface Fuubar {
  void DoStuff()
}

class Fuu : Fuubar {
  Candy candy1
  Wonky wonk
  Candy candy2

  void DoStuff() {

    // these calls must always be done first
    candy1.Do(65)
    wonk.DoStuff("wabba")
    candy1.Do(14)
    candy1.Do(0)
    candy1.Do(1)
    wonk.DoStuff("wibbi")
    candy1.Do(147)

    // these calls are optional and could be different in a sub-class
    candy2.Do(65)
    candy2.Do(14)
    candy2.Do(0)
    candy2.Do(1)
    candy2.Do(147)
    wonk.DoStuff("webby")
  }
}

class Bar : Fuubar {
  Wonky wonk
```

```
  Candy candy2

  void DoStuff() {

    // these calls must always be done first
    candy2.Do(65)
    wonk.DoStuff("wabba")
    candy2.Do(14)
    candy2.Do(0)
    candy2.Do(1)
    wonk.DoStuff("wibbi")
    candy2.Do(147)

    candy2.Do(117)
    candy2.Do(137)
    wonk.DoStuff("jebby")
  }
}

// Developer note: there is a high probability that we need to add more classes
// that realize the interface in the coming sprints

class Wonky {
  void DoStuff(string str) {
    ...
  }
}

class Candy {

  void Do(int number) {
    ...
  }
}
```

```
//
// This code is functional but has a glaring problem
// Your task is to identify the problem and refactor it using
// a standard OO-method. You should update the code and make a
// correct UML class diagram over your solution
//

// example of usage
Fuu f = new Fuu(1, 3)
f.DoStuff()

class Fuu {
  int typeOfAlgorithm1
  int typeOfAlgorithm2

  Fuu(int algo1, int algo2) {
    typeOfAlgorithm1 = algo1
    typeOfAlgorithm2 = algo2
  }

  int DoStuff() {
    int result1;
    int result2;

    switch (typeOfAlgorithm1) {
      case 1:
        result1 = DoAlgorithm1A()
      break;
      case 2:
        result1 = DoAlgoritm1B()
      break;
      case 3:
        result1 = DoAlgoritm1C()
      break;
    }

    switch (typeOfAlgorithm2) {
      case 1:
        result2 = DoAlgorithm2A(result1)
      break;
      case 2:
        result2 = DoAlgoritm2B(result1)
      break;
      case 3:
        result2 = DoAlgoritm2C(result1)
```

```
      break;
    }

    return result2;
  }

  int DoAlgoritm1A() {
    // A massive amount of special code A
    ...
  }

  int DoAlgoritm1B() {
    // A massive amount of special code B
    ...
  }

  int DoAlgoritm1C() {
    // A massive amount of special code C
    ...
  }

  int DoAlgoritm2A(int number) {
    // A massive amount of special code D
    ...
  }

  int DoAlgoritm2B(int number) {
    // A massive amount of special code E
    ...
  }

  int DoAlgoritm2C(int number) {
    // A massive amount of special code F
    ...
  }
}
```

```
//
// This code uses the Model-View-Controller architectural pattern
// but has a glaring problem Your task is to identify the problem
// and solve it using a standard OO-method.
// You should update the code and make a
// correct UML class diagram over your solution
//

model.Foo f = new model.Foo()
view.IView v = new View()
controller.Controller c = new Controller(f, v)

c.Do()

namespace controller {
  class Controller {
    model.Foo model;
    view.IView view;

    Controller(model.Foo m, view.IView view) {
        model = m
        view = v
    }

    void Do() {

      if (v.UserWantsToDoBusinessLogic()) {
        if (m.DoSomeBusinessLogic() != true) {
          v.DrawErrorMessage()
        }
      }
      v.DrawInterface()
    }
  }
}

namespace view {
  interface IView {
     void UpdateProgress(string progressMessage)
     void DrawInterface()
     bool UserWantsToDoBusinessLogic()
     void DrawErrorMessage()
  }

  class View : IView {
```

```
    void UpdateProgress(string progressMessage) {
      // show progress in user interface
    }

    void DrawInterface() {
      // Draw some gui stuff including the results
      ...
    }

    bool UserWantsToDoBusinessLogic() {
        ...
    }

    void DrawErrorMessage() {
      ...
    }
  }
}

namespace model {

  class Foo {

    IView iv;

    Foo() {
      iv = new View()
    }

    bool DoSomeBusinessLogic() {
      ...
      iv.UpdateProgress("0%")
      // some time consuming stuff
      ...
      iv.UpdateProgress("10%")
      // some time consuming stuff
      ...
      iv.UpdateProgress("20%")
      // some time consuming stuff
      ...
      ...
      iv.UpdateProgress("100%")

      return true
    }
  }
```

```
}
```