ENGLISH

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

EXAM IN COURSE
TDT4100 Object-Oriented Programming / IT1104 Programming, Advanced Course

Tuesday 29. Mai 2007
09.00 – 13.00

Contact during the exam:
Trond Aalberg, tlf (735) 9 79 52 / 976 31 088

Allowed books and tools:

- One and only one printed book about Java.

Availability of results:
Results will be available by 19. June 2007.
Results will be available on http://studweb.ntnu.no/ or by phone 81 54 80 14.

Percentages indicates how much each part counts.

Note: All programming must be in Java.

Good luck!

General information

Classes and methods that you may find useful can be found on the last pages of this exam paper (appendix).

Some tasks build upon each other, but in most cases it is possible to solve a following task without having a solution to the first.

PART 1 (25 %): Methods, arrays, control structures

a) Implement the method `boolean inRange(int value, int lower, int upper).` This method can be used to find out whether a value is within the limits lower and upper. E. g. : inRange(4, 2, 6) should return true. The limits should be handled according to these rules: inRange(4,4,4) should return true, whereas inRange(4,5,3) should return false.

b) Implement the method `boolean inRange(int value, int limit1, int limit2).` This method can be used to find out whether a value is within the limits limit1 and limit2. The difference from the previous task is that in this method the highest value from limit1 and limit2 should be used as the upper limit, whereas the smallest value should be used as the lower limit.

c) Implement the method `int[] sortedCopy (int[] tab).` This method should return a sorted copy of the array tab. If the method is called with the array [1,4,3,2] as parameter, it should return the array [1,2,3,4].

d) Implement the method `int findFirstDifferent(int[] tab1, int[] tab2).` This method should return the index of the first position in the array tab1 that contains a value that is different from the corresponding position in the array tab2. If all positions in tab1 have corresponding value in tab2, the method should return -1.

e) A *set* is a collection of unique values (the collection will not contain many of the same value). Implement the method `int[] toSet(int[] tab).` This method takes an array of av `int` as parameter and returns the set of unique values as an array of int. If the method is invoked with the array [1, 3, 5, 6, 5, 1], then it should return the array [1,3,5,6]. The order of the values in the resulting array is not significant.

f) Implement a test-method for testing the method that is described in a). It should be possible to use the method in a subclass of TestCase in the JUnit-framework. Use the methods `assertEquals(Object, Object)` and `assertTrue(boolean)` in TestCase to verify the values.

PART 2 (35 %): Classes

In this part the assignment is to make a class for dates with then name `Date`. Objects of this type hold information about year, number of month (1-12) and number of day (1-31). To keep this simple we assume that all months have 31 days.

a) Implement the class `Date` and the fields/variables that is needed to store the information needed for a data: year, number of month and number of day in month.

b) Implement a constructor that takes year, month and day as parameters. All should be of type int.

c) Explain what encapsulation is and why we use encapsulation.

d) Explain and show how you would encapsulate the class `Date` to enable objects to be immutable. This means that it should not be possible to change the values of an object once it is created.

e) Implement methods for validating the parameter values for the constructor. By using these methods it should be possible to check if the parameters values passed in the constructor are valid. The methods should return `true` or `false`. The methods should be named `validYear`, `validMonth` and `validDay`. The value for year is valid if it is 1–3000. The value for month is valid if it is 1–12. The value for day is valid if it is 1–31.

f) Like any other method the constuctor can throw exceptions. Implement your own exception class named `InvalidDateValue`, and implement a version of the constructor that throws exceptions of this type if it is passed a value that is not valid. Use the methods that are described in e).

g) Implement a toString-method for the class Date that returns the date in the format YYYY-MM-DD.
Explain *why* we are able to use `System.out.println(date)` to print out the value that is returned from the toString-metoden of the date object.

h) What are the characteristics of a method that is declared as `static`?

i) Implement the method `static boolean isValidDateFormat(String date).` This method is used to validate that a date-string is in the format YYYY-MM-DD, and whether the values are according to the rules in e).

PART 3 (15%): Enum

Java supports *enumerated types* by the use of special kind of class called `enum`. This is declared by the use of `public enum Name { ... }`.

a) Describe the characteristics of `enum`, and explain why we sometimes need to use `enum`.

b) Implement an enum named `Month` for months. Show how `Month` can be used to define months including the appropriate name of months (”January”, February”, etc), the correct number for the month (1–12) and number of days in a month (January has 31 days, February has 28, March has 31, etc).

PART 4 (25%): Inheritance, interfaces and cooperation

In this assignment you will write a class name Person. Persons have a name and a phone number. Other persons, such as friends, would of course like to be informed when a persons changes his phone number. Your task is to implement the functionality needed for enabling persons to be informed by other persons updated phone number (e.g. for maintaining a phone book)..

In this task it is required that you use the observed/observable pattern and it is required that you use the class `Observable` and the interface `Observer`. The definition of these is included in the appendix of the exam paper. Assume that the class `Observable` already is implemented.

a) Implement the class `Person` with fields for name and phone number and explain how you would use the class `Observable` and the interface `Observer`. Implement a constructor with a parameter for name and another parameter for phone number (both of type `String`).

b) Implement the method `setPhoneNumber(String number)`. This method should have functionality for notifying all observers whenever a phone number is changed. Note that this method should use methods from `Observable` and `Observer`, and that you may need to implement additional methods.

c) Implement `update(Observable o, Object arg)` and show how you can use this method e.g. to update entries in a person’s phonebook.

d) Draw a sequence diagram showing the sequence of method calls from a person is added as observer and to the person gets information about the update of a phone number.

Appendix

Relevant classes and methods:

Methods in the class String:

```
String(char[] value)
// Allocates a new String so that it represents the sequence of
// characters currently contained in the character array argument.

char charAt(int index)
// Returns the char value at the specified index.

int length()
//Returns the length of this string.

char[] toCharArray()
// Converts this string to a new character array.

String concat(String str)
// Concatenates the specified string to the end of this string

String[] split(String regex)
// Splits this string around matches of the given regular expression.
```

Methods in the class Arrays:

```
static boolean equals(char[] a, char[] a2)
// Returns true if the two specified arrays of chars are equal to one
//another.

static void sort(char[] a)
//Sorts the specified array of chars into ascending numerical order.

static int[] copyOfRange(int[] original, int from, int to)
//Copies the specified range of the specified array into a new array.
```

Methods in the class Integer:

```
static int parseInt(String str)
// Parses the string argument as a signed decimal integer.
```

Methods in the class Character:

```
static boolean isDigit(char c)
// Determines if the specified character is a digit.
```

Observable class

```
public class Observable {

      public Observable()
      //Construct an Observable with zero Observers.

      public void addObserver(Observer o)
      //Adds an observer to the set of observers for this object,
      //provided that it is not the same as some observer already
      //in the set.

      deleteObserver(Observer o)
      //Deletes an observer from the set of observers of this object.

      public protected  void setChanged()
      //Marks this Observable object as having been changed;
      //the hasChanged method will now return true.

      public boolean hasChanged()
      //Tests if this object has changed.

      public void notifyObservers(Object arg)
      //If this object has changed, as indicated by the hasChanged method,
      //then notify all of its observers and then call the clearChanged
      //method to indicate that this object has no longer changed.
      //Each observer has its update method called with two arguments: this
      //observable object and the arg argument.

      protected  void clearChanged()
      //Indicates that this object has no longer changed, or that it has
      //already notified all of its observers of its most recent change,
      //so that the hasChanged method will now return false.

}
```

Observer interface

```
public interface Observer {

      public void update(Observable o, Object arg);
      //This method is called whenever the observed object is changed.
      //An application calls an Observable object's notifyObservers
      //method to have all the object's observers notified of the change.

}
```