ENGLISH

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

EXAMPLE ANSWERS

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

Tuesday 29. Mai 2007
09.00 – 13.00

Please note that the answers in this example answer only show one way to solve the problems.

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 indicate 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.

```
boolean inRange1(int value, int lower, int upper){
      //can be solved in different ways, this is the most compact solution
      return (lower <= value) && (value <= upper);
}
```

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.

```
//You need to find the lowest value and highest value first
boolean inRange2(int value, int limit1, int limit2){
//Using the Math class
      return Math.min(limit1, limit2) <= value &&
            value <= Math.max(limit1, limit2);
}

boolean inRange4(int value, int limit1, int limit2){
//using your own test
if (limit1 <= limit2){
      return (limit1 <= value) && (value <= limit2);
}else{
      return (limit2 <= value) && (value <= limit1);
}
}
```

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].

```
//Important to return a copy!
//Sorting e.g. by the use of Arrays.sort() or user defined sort-method

int[] sortedCopy(int[] tab){
      int[] copy = new int[tab.length];
      for (int i = 0; i <= tab.length; i++){
            copy[i] = tab[i];
      }
      Arrays.sort(copy);
      return copy;
}
```

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.

```
//Use for or while to iterate
//Test for equal/not equal values
//Find shortest array to prevent IndexOutOfBoundsException


int findFirstDifferent1(int[] tab1, int[] tab2){
      int i = 0;
      int length =
            (tab1.length <= tab2.length ? tab1.length : tab2.length);
      while (i < length && tab1[i] == tab2[i]){
            i++;
      }
      if (i < length){
            return i;
      }else{
            return -1;
      }
}

int findFirstDifferent2(int[] tab1, int[] tab2){
      int length =
            (tab1.length <= tab2.length ? tab1.length : tab2.length);
      for (int i = 0; i < length; i++){
            if (tab1[i] != tab2[i]){
                  return i;
            }
      }
      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.

```
int[] toSet(int[] tab){
      //Create a new array of the same size (or use ArrayList)
      //Test if the value  at position p exists in the remaining
      // of the array (p+1 – tab.length)
      //if not copy the value
      //Finally, make a copy of the array with the relevant range
      int set[] = new int[tab.length];
      int pos = 0;
      for (int i = 0; i < tab.length; i++){
            if (! isInArray(tab, tab[i], i + 1, tab.length)){
                  set[pos++] = tab[i];
            }
      }
      return Arrays.copyOfRange(set, 0, pos);
}

//Utility-method for above method (or the same loop can be included in
//the method above


public boolean isInArray(int[] tab, int value, int from, int to){
      boolean found = false;
      for (int i = from; i < tab.length; i++){
            if (value == tab[i]){
                  found = true;;
            }
      }
      return found;
}


//Alternative solution using Collection API
int[] toSet1(int[] tab){
      Set<Integer> set = new HashSet<Integer>();
      for (int i : tab)
            set.add(i);
      int[] settab = new int[set.size()];
      int counter = 0;
      for (int i : set)
            settab[counter++] = i;
      return settab;
}



//Another solution is to use ArrayList and the contains-method
```

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.

```
// Text is a bit misleading, assert-methods are only examples and it is
// not a requirement to use the mentioned methods.
// For all validations you can use assertTrue(), with not(!) if
// you need to test for false (or use assertFalse()).
// Assuming that the inRange method is a static method

public void testInRange(){
      // Assuming that the inRange method is a static method
      // Testing for true if the value is within the range
      assertTrue(Methods.inRange(4, 2, 6));
      // Testing for false if the value is not within the range
      assertTrue(!Methods.inRange(6, 1, 5));
      //Testing correct  behaviour for limits
      assertTrue(Methods.inRange(4, 4, 4));
      assertTrue(!Methods.inRange(6, 1, 5));
}
```

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 are needed to store the information needed for a date: year, number of month and number of day in month.

```
public class Date {

      //modifier is not specified yet,
      //but later it is required to use private
      private int year;
      private int month;
      private int day;

}
```

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

```
public Date(int year, int month, int day){
      this.year = year;
      this.month = month;
      this.day = day;
}
```

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

Encapsulation is to hide the fields (using the modifier protected) and use methods to access and change the values of an object. Encapsulation is used to protect the values of objects and to enable that the internal of an object can be changed without having to change the publicly accessible methods.

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.

By using the modifier private for the fields, implementing a constructor to instantiate the objects with correct values and use getMethods to access the data. By *not* implementing setMethods there is no way to change the state of an object once it is created.

```
public class Date {

      private int year;
      private int month;
      private int day;

      public Date(int year, int month, int day){
            this.year = year;
            this.month = month;
            this.day = day;
      }
      public int getYear(){
            return year;
      }

      public int getMonth(){
            return month;
      }

      public int getDay(){
            return day;
      }

}
```

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.

```
public static boolean isValidYear(int year){
      return inRange(year, 1, 3000);
}

public static boolean isValidMonth(int month){
      return inRange(month, 1, 12);
}

public static boolean isValidDay(int day){
      return inRange(day, 1, 31);
}
//If you are using these methods in 2i) they have to be defined as static
//or you need to explain in 2i) that they should have been static
//Rather than repeating the same kind of test
//it is an advantage to mage a separate method for testing
//comparable to the method described in part 1 a)
private static boolean inRange(int value, int lower, int upper){
      return (lower <= value) && (value <= upper);
}
```

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).

```
public class InvalidDateValue extends Exception{
//a better name for this class would be InvalidDateValueException, but
//this is a long name when writing by hand ;-)

      public InvalidDateValue(){ }

      public InvalidDateValue(String msg){
            super(msg);
      }

}


//Constructor in Date

public Date(int year, int month, int day) throws InvalidDateValue{
      if (isValidYear(year) && isValidMonth(month) && isValidDay(day)){
            this.year = year;
            this.month = month;
            this.day = day;
      }else{
            throw new InvalidDateValue("Invalid date format");
      }
```

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.

```java
public String toString(){

      String yearpadding = "";
      if (year < 1000)
            yearpadding += "0";
      if (year < 100)
            yearpadding += "0";
      if (year < 10)
            yearpadding += "0";

      String monthpadding = "";
      if (month < 10)
            monthpadding += "0";

      String daypadding = "";
      if (day < 10)
            daypadding += "0";

      return yearpadding + year + "-" +
                  monthpadding + month + "-" + daypadding + day;

      // or use of the format-method in the String class
      // (not covered in the curriculum)
      // return String.format("%04d-%02d-%02d", year, month, day);

}
```

All classes in Java inherit from the built-in class Object. This class defines a default toString()-method that all subclasses inherit.
When we write our own toString mentod with the same signature, our method overrides the toString-method of the Object class: it is the toString-implementation of the actual class that is invoked (or the implementation of the nearest parent if the class itself does not reimplement this method). The println-method takes as parameter an Object-type and invokes the toString-method, which then according to the above causes the toString-method we have implemented to be invoked.

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

```
Static methods are class methods and can be invoked by the use of
ClassName.method(). Static methods only run in the context of class-
variables: can only access variables and other methods that are declared
as static. Static methods are typically used for utility-functions that
are independent of a particular state of an instance.
```

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).

```java
public static boolean isValidDateFormat(String date){

      String parts[] = date.split("-");

      //the date should have 3 parts
      if (parts.length != 3)
            return false;

      //year should be 4 characters,
      //month and day should be two characters
      if (parts[0].length() != 4)
            return false;
      if (parts[1].length() != 2)
            return false;
      if (parts[2].length() != 2)
            return false;
      //characters should be  digits
      for (int i = 0; i < parts.length; i ++){
            if (! isDigit(parts[i]))
                  return false;
      }
      int year = Integer.parseInt(parts[0]);
      int month = Integer.parseInt(parts[1]);
      int day = Integer.parseInt(parts[2]);

      //int values should be within the defined ranges
      if (isValidYear(year) && isValidMonth(month) && isValidDay(day)){
            return true;
      }else{
            return false;
      }
}

public static boolean isValidYear(int year){
      return inRange(year, 1, 3000);
}
//also applies to isValidMonth, isValidDay, and inRange
```

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`.

```
Enum is a specific kind of class with a predefined set of ordered, named
instances. At runtime we are not able to create new instances of enums but
can refer to instances in the set by their name. Enum can have
constructor, fields and methods just as ordinary classes. We typically use
enum when we want to make sure that a only a fixed set of named instances
exists, such as when we need a predefined and fixed number of named things
(months, colors, etc).
```

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).

```java
public enum Month {

	JANUARY("January", 31), FEBRUARY("February", 28),
	MARCH("March", 31), APRIL("April", 30),
	MAY("May", 31), JUNE("June", 30),
	JULY("July", 31), AUGUST("August", 31),
	SEPTEMBER("September", 30), OCTOBER("October", 31),
	NOVEMBER("November", 30), DECEMBER("December", 31);

	// Enum-names is often in uppercase by convention
	// To be able to return correctly capitalized names we may
	// use a separate field for this
	// (but this can be solved in different ways)

	private int days;
	private String name;

	Month(String name, int days){
		this.name = name;
		this.days = days;
	}

	public int getDays(){
		return days;
	}

	public int getNumber(){
		return this.ordinal() + 1;
	}

	public String getName(){
		return this.name;
```

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)`.

```
Persons have to be both Observers and Observable. The Person class has to
extend the class Observable and implement the interface Observer. The
class inherits methods from Observable but the methods declared in the
interface Observer must be implemented in the Person class.


public class Person extends Observable implements Observer{


      private String phonenumber;
      private String name;

      public Person(String name, String phonenumber){
            super();
            this.name = name;
            this.phonenumber = phonenumber;
      }

      public void update(Observable p, Object n) {
            // Do something when notified of a change
      }
}
```

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.

```
//This is the most simple solution

public void setPhoneNumber(String phonenumber){
      this.phonenumber = phonenumber;
      setChanged();
      this.notifyObservers(this.phonenumber);
      // According to the description the notifyObserver-method has
      // an implicit call to clearChanged().
}
```

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.

```
//Assuming that we have a phonebook based on java.util.Map
//private Map<String, String> phonelist;
//You should check with instanceof that it is a person-object
//The arg-variable can be any type of object, but in this example we
//assume that it is a phonenumber as a String
public void update(Observable p, Object n) {
      if ((p instanceof Person) && (n instanceof String)){
            Person person = (Person) p;
            String phone = (String) n;
            phonelist.put(person.getName(), phone);
      }
}
```

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.

```
In the answer you should a) show that you know what a sequence diagram is
and b) show the correct sequence of method-invocations between a main-
method, person objects a and b.
a.addObserver()
a.setPhoneNumber()
a.setChanged()
a.notifyObservers()
b.update()
a.clearChanged()¨
and possible other methods that are invoked e.g. in b.update
```

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.

}
```