Mittuniversitetet
MID SWEDEN UNIVERSITY

Examination

Java for C++-programmers

Distance courses from Östersund (DSV)
2018-01-19, kl. 9.00 – 21.00 (12 hours, Swedish time)

Number of tasks: 5

Grading :

F	Fx	E	D	C	B	A
Not executable solution	Partially solved task 1	Approved solution for task 1	Approved solution for task 2	Approved solution for task 3	Approved solution for task 4	Approved solution for task 5

Grade E-A means pass, Fx and F means fail. For grade E, an approved and executable solution for task 1 is required. For grade D, an approved and executable solution for task 1 and 2 is required and so on…

Instructions:

In addition to the rules specified in Moodle, for this course the following applies:

- The exam will be written personally. It is not allowed to copy code from anyone else, from web pages or some other source like that. It is also not allowed to take help from any other person.
- It is allowed to use literature, APIs, own code from labs, and to search information on the web.
- Before writing time is finished, you must submit your exam. This will be done in Moodle in the same way and in the same place where you have previously submitted your assignments.
- You will submit a zip file containing all files you have written to solve the tasks (no .class files). If you have used Eclipse to solve the tasks, you will pack the entire project into a zip file. Name the zip file as follows:
  ååmmdd-nnnn, Efternamn Förnamn.zip
- The exam will be written in the location you specified in your registration.
- If there are any problems with your network connection during the exam period or when you submit your answer, you should contact
  Robert Jonsson, robert.jonsson@miun.se
  Awais Ahmad, awais.ahmad@miun.se

Introduction

Read all the tasks before you start writing code. If you aim for one of the higher grades, you should consider this already in the code of the initial tasks. Please don't write all code in one and the same class, divide the code into appropriate classes and methods. For each new task , it is suggested to copy the entire previous task to a new folder so that the previous task remains in the unchanged state. Submit only the solution for the grade you're aiming for. You can reuse or build on code from your assignments or examples in the course book.

Write a comment about the highest grade/assignment you attempted or solved while submitting you exam in Moodle in the comments section.

In tasks where a GUI is used, it is not allowed to notify the user of what is happening in the application with System.out.print (however, debug printouts are allowed). In a GUI, it is also not allowed to retrieve input data from the user with e.g. Scanner or System.in. If you are considering anything or something unclear about the exam, just contect us via e-mail or put a question on course forum.

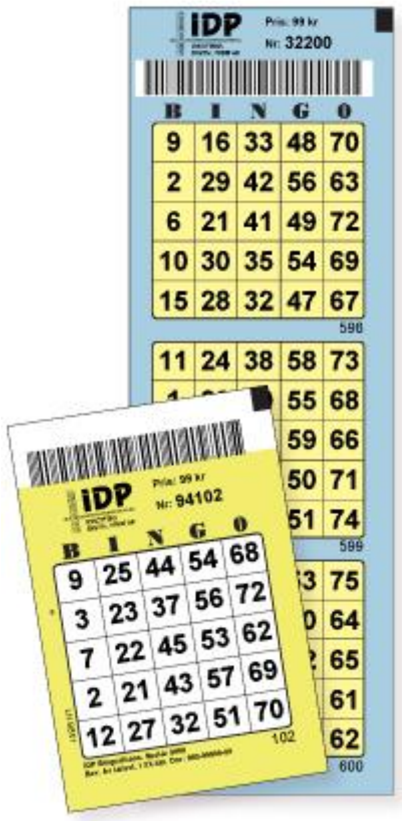
Making minor mistakes and errors in a task does not necessarily mean that the task is not approved. However if the multiple tasks contains errors and misses, the grade may be lowered by one or more steps. For example, if you have submitted a solution for task 4 (grade B) and tasks 1 and 3 contain some mistakes, your grade may be lowered to C.

This exam is about a game Bingo.

Task 1 (grade E)

Bingo is a game where a random announcer calls a name (Bertil, Ivar, Olof, Niklas or Gustav) followed by a number (1 - 75), then the participants mark the numbers on their card if those numbers are also available on the card. When an entire row will be selected vertically , horizontally or diagonally, the player will get a bingo.

Each card consists of five vertical columns marked **B**, **I**, **N**, **G** and **O**. Column **B** contains numbers between 1 and 15, column **I** contains numbers between 16 and 30 and so on. A classic bingo card contains 25 numbers (five columns and five rows) and may look like the front card in the image below. There is also a variant in which the tray contains all 75 numbers divided into 3 groups of 25 numbers (see the rear card in the figure below). The numbers are randomly placed in each column and each number is unique in one card (is available only once in one card).



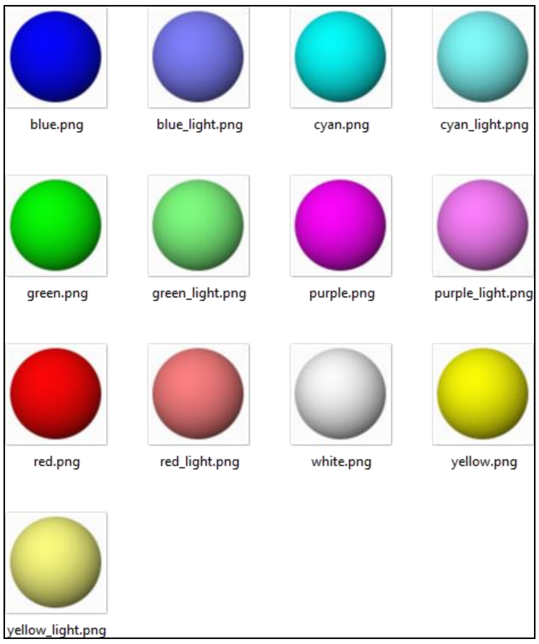
In the first task, you will create a your own graphic competent (class that inherits `JComponent` or `JPanel`) to serve as "announcer" in the bingo game. The component will present the bingo numbers drawn for the user (randomly selected). In the component, at least the last three drawn bingo numbers should appear at the same time (preferably more numbers, for example by scrolling back and forth). The component should be written in a class called `BingoCaller`.

By bingo numbers mean the following:

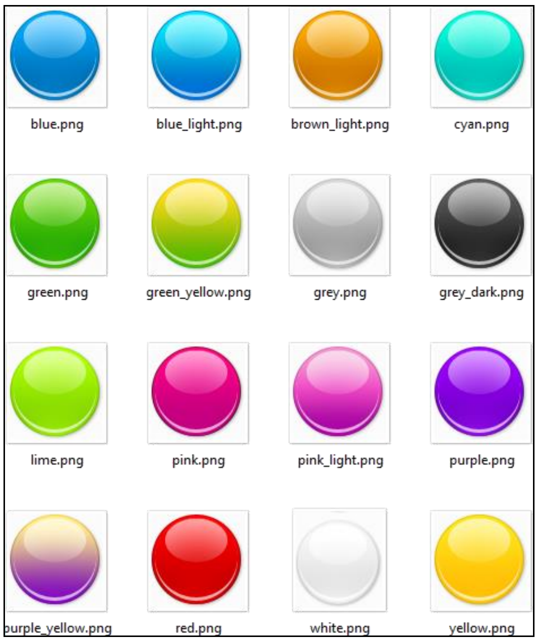
Nummer 1 – 15 gives bingonummer B1, B2, B3, …, B15.
Nummer 16 – 30 gives bingonummer I16, I17, I18, …, I30.
Nummer 31 – 45 gives bingonummer N31, N32, N33, …, N45.
Nummer 46 – 60 gives bingonummer G46, G47, G48, …, G60.
Nummer 61 – 75 gives bingonummer O61, O62, O63, …, O75.

For example, for number 7,your `BingoCaller` should show B7. For number 55, the component will display the G55. And so on. Bingo numbers should be represented by a class called `BingoBall`.
In this task the component `BingoCaller` do not need to randomize numbers, it is enough that it can show the bingo numbers that you enter directly in your code.

Choose your design for the Bingo Ball by yourself. With the exam you will find two zip files containing images of balls you will use in your solution (see Figure 1 and Figure 2). On every ball, bingo number must be printed.



Figur 1



Figur 2

Skapa en klass `BingoGUI` som ärver `JFrame` och som lägger till komponenten överst i fönstret och lägg sen till minst fem bingonummer. I kod kan det t.ex. se ut så här:
The code may look like this:

```
BingoCaller caller = new BingoCaller();
caller.addBingoBall(new BingoBall(1));
caller.addBingoBall(new BingoBall(31));
caller.addBingoBall(new BingoBall(41));
caller.addBingoBall(new BingoBall(61));
caller.addBingoBall(new BingoBall (18));
add(caller, BorderLayout.NORTH);
```

The result of the above code should look like that in Figure 3.

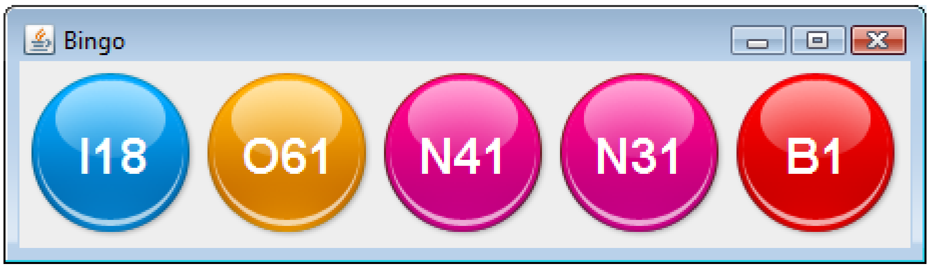


Figure 3

Divide your code in the appropriate methods and classes. Name the class `Bingo` with which you start the application. Other classes will be included in your solution are `BingoBall`, `BingoCaller`, `BingoGUI` and `Bingo`.

When you're done, create a jar file of all the necessary files needed to run the application. Jar file should be name to `bingo.jar` (you do not need to create a jar file now if you want to continue to the next task).

Task 2 (grade B)

Now you will make it possible for the user to create bingo cards and save them to a file. You can decide either it will be 25 or 75 numbers card. Whatever card you choose, the first line should contain the letters B, I, N, G, O, and then the lines after it will be filled with random numbers. You can design bingo card by using signs “|” and “-”(see example below on left) or you can use “*”(See example below on right). You can use other signs as well, if you want.

```
--------------------------          **************************
| B  | I  | N  | G  | O  |          * B  * I  * N  * G  * O  *
--------------------------          **************************
|  6 | 18 | 31 | 50 | 73 |          *  6 * 18 * 31 * 50 * 73 *
--------------------------          **************************
| 14 | 29 | 38 | 59 | 66 |          * 14 * 29 * 38 * 59 * 66 *
--------------------------          **************************
|  5 | 20 | 40 | 51 | 72 |          *  5 * 20 * 40 * 51 * 72 *
--------------------------          **************************
| 13 | 25 | 44 | 46 | 71 |          * 13 * 25 * 44 * 46 * 71 *
--------------------------          **************************
|  1 | 16 | 42 | 54 | 62 |          *  1 * 16 * 42 * 54 * 62 *
--------------------------          **************************
```

From the previous task, you will add a menu option in your `JFrame` that creates and saves a bingo card to a file on the user's hard disk. The Idea is that, the user should be able to print the card to use in the bingo game. Each time when the user chooses to create a bingo card, new numbers should be randomly selected. You decide by yourself how bingo trays will be named (file name).

When you're done, create a jar file of all the necessary files needed to run the application. Jar file should be name to `bingo.jar` (you do not need to create a jar file now if you want to continue to the next task).

Task 3 (grade C)

In this task you will extend the component `BingoCaller` from task 1 with start and stop methods. When the start is called the first number will be displayed, then wait for a while and will show another the number and then wait for a while again and show third number and so on until all 75 numbers have been presented (or until the stop method is called). You can decide by yourself, how long the delay between each draw of the new number should be. In this task you don’t have to handle restarts (i.e. start drawing again after the stop has been selected).

The component can keep track of things, for example, all the numbers, in which order they should be randomized, have a method to display the next number, etc. You can take help from other classes if you want (read the last task).

In your `JFrame` there should be a menu option for starting the draw of all 75 numbers, as well as a menu option that stops the drawing of numbers (see example in Figure 4 and 5).

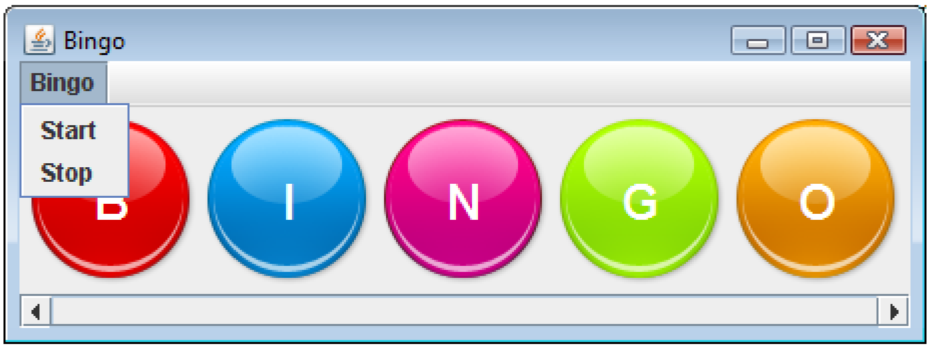


Figure 4

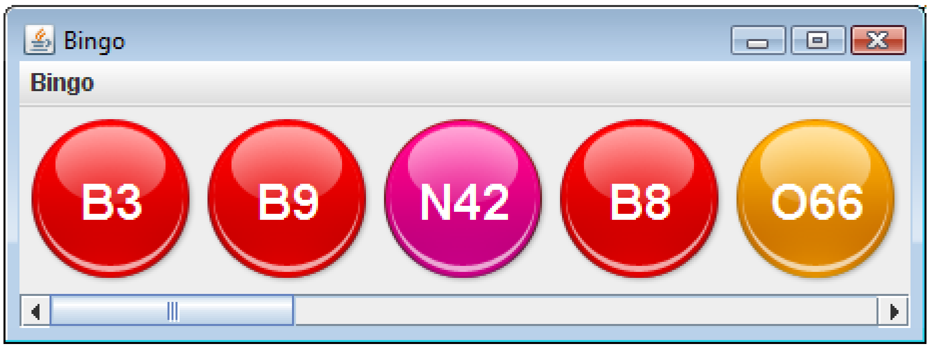


Figure 5

When you're done, create a jar file of all the necessary files needed to run the application. Jar file should be name to `bingo.jar` (you do not need to create a jar file now if you want to continue to the next task).

Task 4 (grade B)

Now it’s time to make a client-server solution for the bingo game. You will create a server that draws bingo numbers and sends them to a client that shows the numbers in its `BingoCaller`. You can choose if you want to use `MulticastSocket` or `Socket` in your solution.

When the client press the menu option ***Start***, the server will randomize new numbers and then send the bingo numbers one by one (with a delay of your choice) until the client press menu option ***Stop*** or until the server sent all 75 numbers.

In this task the server does not have to manage multiple clients simultaneously. And you don’t have to handle restarts(i.e. start drawing again after the stop has been selected).

When you're done, create a jar file of all the necessary files needed to run the application. Jar file for server part should be named to `server.jar` and jar file for client part should be named to `client.jar` (you do not need to create a jar file now if you want to continue to the next task).

Task 5 (grade A)

In the last task you will rewrite the server part so that it can handle multiple users simultaneously. All the user connected to the server should get the same bingo numbers. When the first client, press the Menu option Start your server should start sending numbers to the clients. You can decide by yourself how you want to handle the clients joining late in the game (i.e. clients connecting after the draw of numbers has started).
In the client side you will change menu option ***Stop*** to ***Bingo***. When a user gets bingo, he will press the menu option *Bingo* and the server will stop to draw new number. Server should simultaneously notify connected clients that a user have got bingo and that you can now start a new game by selecting menu option Start. Use for example a dialog box in the client.

When you're done, create a jar file of all the necessary files needed to run the application. Jar file for server part should be named to `server.jar` and jar file for client part should be named to `client.jar`