CSC115 Assignment 1:
Bingo

Objectives

Upon completion of this assignment, you need to be able to:

- Write Java code following the Coding Conventions defined for the course.
- Define and create a simple `class` in Java.
- Create and use Java arrays that contain complex data types (objects).
- Have been introduced to Java **Exception Handling**.
- Use a developed testing system.

Introduction

In the game of Bingo, the *caller* randomly selects a set of Bingo balls, one at a time, from a Bingo cage and calls out the letter and number printed on the little ball. Each player checks for that particular number on their Bingo *card*, a 5×5 grid. If the number is there, then the player will cover the specific grid that contains the number. The player's goal is to be the first to cover the the grids in a previously-agreed-upon pattern, often a straight vertical or horizontal line.

Immediately after the caller's announcement of the current ball, a player who has the necessary pattern will call out "Bingo!". Before claiming the prize, the player proves the validity by calling out the numbers of each of the covered squares that make up the pattern. The caller verifies that each of the balls were actually called, by searching through the *called* balls.

In this assignment, you will create a class `BingoBall` and create the data structure that contains the `called` balls.

Quick Start

(1) Create a fresh directory to contain this assignment: `CSC115/assn1` is a good name. Download this pdf file and the file `BingoBall.java` into that directory.

(2) Create a subdirectory, called `docs`, for *documentation* and download the `html` files. When you open the local version of this pdf, and click on the following links, the *specifications* for the required Java classes will open. Try it:

- BingoBall specifications
- BingoCalls specifications

If you are not using a local version of this document and/or have not created a subdirectory called `docs` to store the html files, you can open one of the html files by double-clicking on it.

(3) Carefully examine both of the specification documents. Note the similarity to the standard documentation pages for all Java classes. For example, search for `java` and `Random` and note the layout of information.

(4) Compile `BingoBall.java`. It compiles and runs without errors, but it is not very useful in its initial condition.

(5) Open `BingoBall.java` in a text editor. A basic template has been started, to guide you through the process of creating a class by taking the following recommended steps:

(a) Create the shell of the program (done for you): This is done by creating the class definition and all the public methods as directed in the specification document. The methods are empty, except when they must return a value, in which case a dummy value is returned. Once the shell is created, the source code can be compiled.

(b) Decide what the private data fields will be: Note that for `BingoBall` a `public static final` array has been initiated for you. This is NOT a data field, but a constant value that you may find helpful. (See the comments in the source code).

(c) Implement the methods one at a time, starting with the easiest ones.

Note that the `main` method contains a set of statements that tests the code. You may add more tests if you like.

(6) After completing and thoroughly testing `BingoBall`, create the `BingoCalls` class, again following the specifications. Create your own `main` method in this class to thoroughly test all the methods in `BingoCalls`.

Detailed Instructions

Complete both the `BingoBall` and `BingoCalls` classes adhering to the following guidelines:

- The associated html file contains a summary and detailed explanation of what each method must do, along with details about any input or returned value. Note that the information provided is generated from header comments; you are welcome to use these as required header commenting within your code.
- The `main`[1]method is to be used as a tester. The `BingoBall` class `main` method is written for you.

  Use this as the standard for testing all of your assignments.
- Compile and test your work frequently. Experienced programmers all do this, knowing that a single syntax error or logical error is easier to fix when you've only made a few changes to the document.

Some basic information **(for details, see the textbook)**

By convention, Java uses very specific names for many of the basic methods, making it easier for users to remember what methods are available. For instance, every class in Java has both an `equals` and a `toString` method. A basic class definition in Java will generally contain the following:

- **private data fields**: These specifically define the attributes that make a particular object of this class unique.
- **One or more constructors**: The constructors are created to allow the user to initialize an object. In most cases, the constructor sets the values of the data fields, as specified by the argument list.

[1]Note to C programmers: unlike C, which requires only one main method for all of the files involved in the running of a program, Java allows for each file to contain a main method. The actual main method that *runs* the program is determined by the `java` command which names the file.

- **Accessor–modifier methods**: These methods are also referred to as the *setters* and *getters*. Where the user is permitted, each of the private data fields can be updated or retrieved using these methods.
- **A method to determine equivalence between objects of the class**: It is always useful to be able to determine what conditions are necessary for two objects of the same class to be deemed equivalent. With some objects, like strings or number objects, this is intuitive. When it is not, the class designer must make the determination. Generally, the combined equality of the data fields is a safe determination, but that may not always be the case.
- **A method to format an informative String that describes the object**: This is an extremely useful method, both during development as a debugging tool, and also for users to visualize the details of the objects.

BingoBall

The `BingoBall` consists of

- An uppercase letter, one of {B, I, N, G, O}.
- A number between 1 and 75. There is a matching of each of the letters to the numbers that can be associated with it.

  B: {1–15}

  I: {16–30}

  N: {31–45}

  G: {46–60}

  O: {61–75}

Some sample testing has been provided for you in the main method of the `BingoBall` class. Note that the `System.out.println` statements with the `BingoBall` argument will produce the output that is defined in the `toString` method. If the `toString` method has not been created, then the address location of the reference is printed instead, which is not very helpful.

BingoCalls

The `BingoCalls` class is basically a means to store the BingoBalls that were called during a particular game. There will only be one instance of this class, as opposed to the 1–75 BingoBalls that can be instantiated. The main data field of this class will be an `array` of BingoBalls. You may use a single array or a 2D array, or an array of arrays for this data field. You MAY NOT use any of the collections data structures that extend the `List` class from the `java.util` package.

In the design, the public methods allow the caller (*the user*) to

- create an empty container at the start of a game,
- add balls as they are called in the game,
- determine how many balls have been played,
- find and remove a ball that was accidentally added (mistakes can happen).
- confirm that a particular ball was played during the game, and
- empty the container of all balls, making it ready for a new game.

Java array data types must have a fixed size that is determined when the array is initialized. (Note that the array must be initialized separately from the individual items inside the array.) To allow for re-usability, we add a provision that doubles the size of the array when the number of items exceeds the capacity of the array. To test for this, create the initial array to hold a maximum of 5 items. Before each insert, make sure that the array is not already full; if it is, then call a private method, created by you, that enlarges the array before inserting the item.

As with the `BingoBall`, thoroughly test everything within the main method of the class.

Submission

Submit the following completed files to the Assignment folder on conneX. You will be shown how to do this in the first week of labs:

- `BingoBall.java`
- `BingoCalls.java`

Please make sure you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send [`.class`] (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission, but only if we know that it's there.

A note about academic integrity

It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

Grading

Marks are allocated for the following:

- Proper programming style is demonstrated, as per the coding conventions on CSC115 conneX Resources.
- All methods are implemented exactly to the specifications.
- The main data field of `BingoCalls` must be an array, and not an object of any of the `List` classes in the Java API, `java.utils` package.
- The private method that resizes the array in `BingoCalls` must copy the contents of the original array one by one into the new larger array.
- The `main` method of `BingoCalls` must demonstrate that each method in `BingoCalls` was tested.

You will receive no marks for any Java files that do not compile.