CS1027

ASSIGNMENT 1

Computer Science Fundamentals II

This assignment is due on Wednesday, June 23, 2021 by 11:55pm. See the bottom for submission details.

Learning Outcomes

To gain experience with

- Creating classes with simple methods
- Looping through 2D arrays
- Using inheritance and overriding methods

Introduction

A new city is being developed on an island and you are hired to help with the city planning! Your computer science skills are needed to develop a program to represent the potential layout of the new city and determine which buildings can be constructed and which locations are valid for those buildings. Use a 2D array to represent the city layout and insert the various Building objects into the corresponding cells of the array where the building is being added, but only if the proposed location is a valid place for it!

The standard rule for placing buildings is that they cannot stretch out of the bounds of the city and that a building can not overlap with any other building. There are 2 sub-types of buildings, Marina and Skyscraper, which each have their own placement rules in addition to the standard building rule. Marinas must be touching the water's edge so you will have to check that at least one cell of a Marina is along an edge of the city. Skyscrapers have an additional property (instance variable) for their height. The height of a Skyscraper must be less than 10 units and it must be greater than or equal to the base area (width * length). For example, a skyscraper whose base is 2 by 3 can have a height of 6, 7, 8, or 9. Both Marina and Skyscraper must be child classes of the Building class using inheritance.

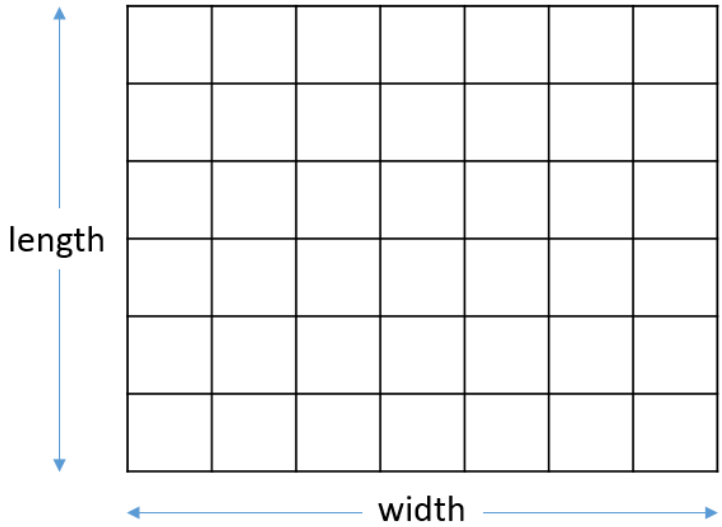

Figure 1. An empty city of width 7 and length 6. Assume we are looking straight down at the city from a helicopter to see the layout like this. Note the use of width and length for this assignment.

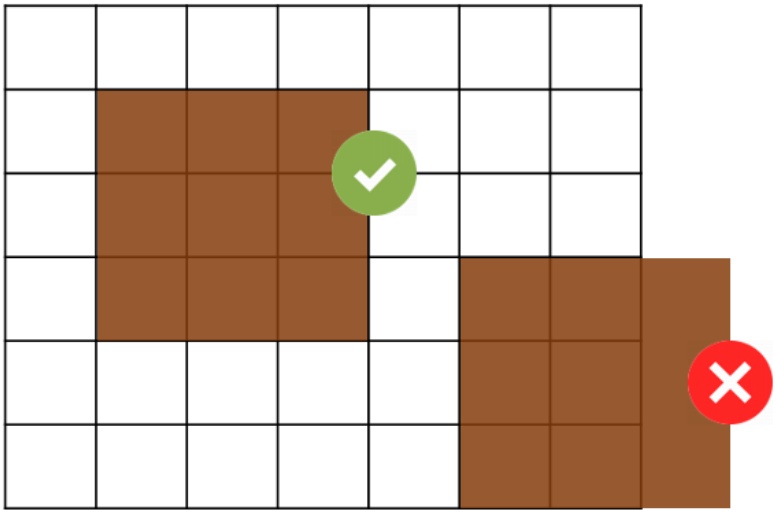

Figure 2. An example of one valid placement for a building (upper-left) and one invalid placement for a building (lower-right). This is invalid because the building extends beyond the bounds of the city.

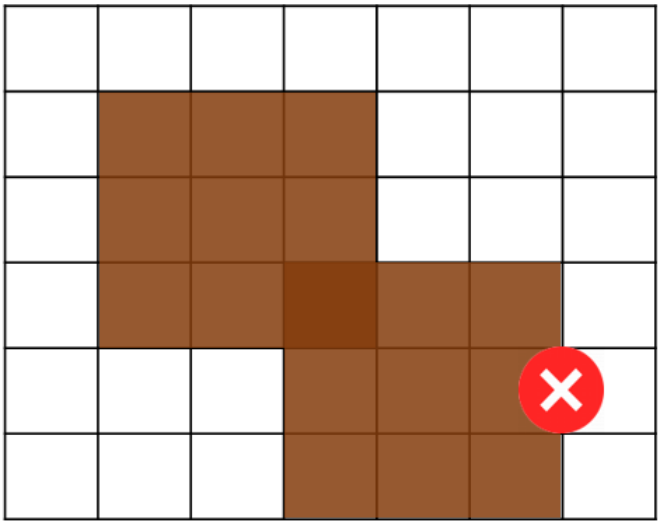

Figure 3. An example of an invalid placement for a building (lower-right) after already placing a valid building (upper-left). The new building is invalid because it overlaps with the existing building. On its own, the placement would be valid, but it is invalid due to the overlapping.

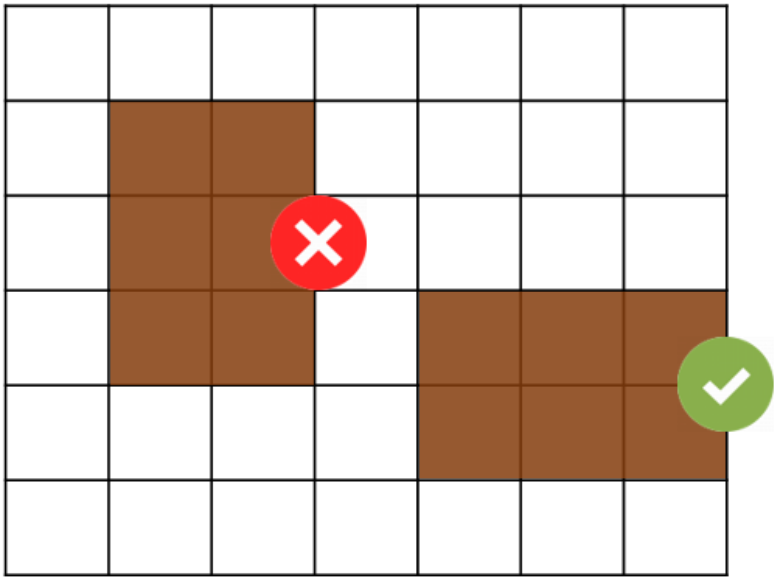

Figure 4. An example showing a valid placement of a Marina (right) and an invalid placement of a Marina (left). This is invalid because it does not touch the edge of the city in at least one cell.

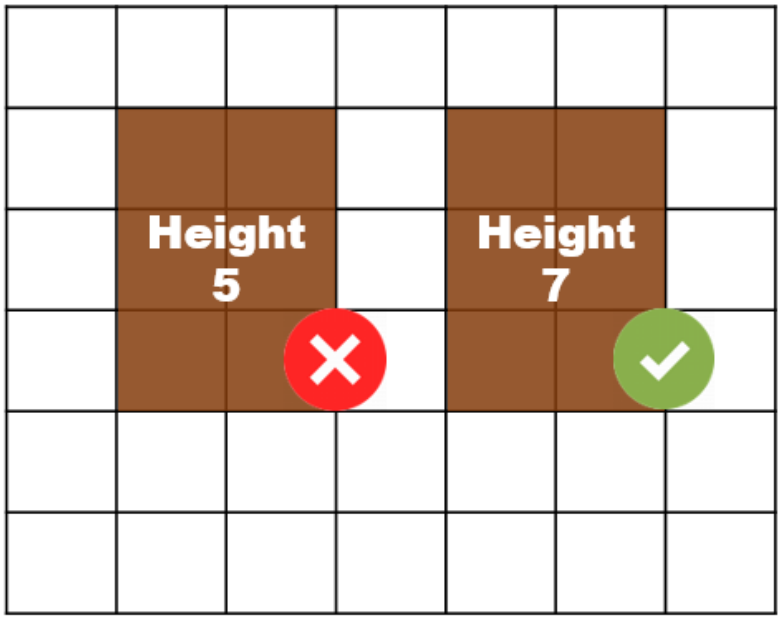

Figure 5. An example of a valid Skyscraper (right) and an invalid Skyscraper (left). This is invalid because the height is 5 which is less than its base area (2 x 3 = 6). The height must be at least as large as the base area but less than 10.

Provided files

The following is a list of files provided to you for this assignment. Please do not alter these files in any way.

- TestCity.java
- TestInheritance.java

Classes to implement

For this assignment, you must implement 4 Java classes: *Building*, *City*, *Marina*, and *Skyscraper*. Follow the guidelines for each one below.

In all of these classes, you can implement more private (helper) methods, if you want to, but you may **not** implement more public methods. You may **not** add instance variables other than the

ones specified below nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

Building.java

This class represents a building structure to be placed within the city. Note that the Marina and Skyscraper classes (explained below) will be sub-classes of this Building class.

The class must have the following *private* variables:

- symbol (String)
- width (int)
- length (int)

The class must have the following *public* methods:

- public Building(String, int, int) [constructor]
  - Initialize each of the instance variables (symbol, width, length) with the corresponding parameters
- public int getWidth()
  - Returns the width
- public int getLength()
  - Returns the length
- public boolean isValidPlacement(Building[][], int, int)
  - Takes in a 2D Building array (representing the city – see below for more details) and the x and y positions representing the location where we will attempt to place this building.
  - Use the rules for a standard building described near the top of this document (the building cannot reach outside the city bounds and cannot overlap with another building in any cell) to check if it can be placed in the given (x,y) position.
  - Return true if the building can be placed there; otherwise return false
- public String toString()
  - Simply return the symbol variable to represent the building

City.java

This class represents the new city that is being developed.

The class must have the following *private* variables:

- width (int)

- length (int)
- layout (Building[][])

The class must have the following *public* methods:

- public City(int, int) [constructor]
  - Takes in parameters for x and y and assigns them to the corresponding instance variables.
  - Initialize the layout 2D array using length and width as the array sizes (note that the order is important for which array is initialized with which dimension value)
- public boolean addStructure(int, int, Building)
  - Takes in x and y position values and a Building object that we are attempting to add to the city at the given (x, y) position.
  - Call the isValidPlacement() method on the given Building object (remember that since we will be creating sub-classes of Building, the isValidPlacement() method might be the original or it might be the overridden method from the corresponding sub-class). If this method returns false, simply return false from this method too.
  - If the isValidPlacement() method returns true, we can add the Building object to that location so loop through ALL the cells in which the Building will be constructed (not just the top-left corner cell of the Building's location) and point each of those cells to the given Building object. Then return true.
- public String toString()
  - Loop through the layout 2D array and construct a string with all the values in a well-formatted grid. Any cells that do not contain a Building object should be shown with a period (.) and the ones that do contain a Building should be shown with that Building object's toString() (which is typically its symbol).
  - Add 2 spaces after each cell's symbol, and add a newline character at the end of each line so that the string is formatted like a grid.
  - Return the string that contains this entire grid-like representation of the layout.

Marina.java

This class is one of the sub-classes of Building, so it must be declared as such at the top. This one is a Marina, so it must be adjacent to "water", meaning at least one cell of this structure must be along the edge of the city.

The class should not have any new (other than the inherited) instance variables:

The class must have the following methods:

- public Marina (String, int, int)
  - Takes in the same parameters as Building. Call super() with these values.
- public boolean isValidPlacement(Building[][], int, int)

- Checks to see if this building meets the placement requirements for a Marina (it must be touching the edge of the city in at least one cell) AND the original Building requirements (hint: call super.isValidPlacement() to access the Building's isValidPlacement method) at the given (x,y) position
- Returns boolean to indicate whether or not this Marina can be placed there

Skyscraper.java

This class is the other sub-class of Building. This one represents a Skyscraper building and it has a new instance variable to represent the height of the skyscraper. The placement rules specific to this type is based on the height. The height must be less than 10 and it must be greater than or equal to the base area (width x length).

The class must have the following *private* variable:

- height (int)

The class must have the following methods:

- public SkyScraper(String, int, int, int) [constructor]
  - The first 3 parameters in this constructor are the same 3 from the superclass constructor (symbol, width, and length). The 4th one here is for the height. Set this value to the new instance variable after calling the super's constructor.
- public boolean isValidPlacement(Building[][], int, int)
  - Checks to see if this building meets the placement requirements for a Skyscraper (its height must be less than 10 and greater than or equal to its base area) AND the original Building requirements (hint: call super.isValidPlacement() to access the Building's isValidPlacement method) at the given (x,y) position
  - Returns boolean to indicate whether or not this Skyscraper can be placed there
- public String toString()
  - Override the superclass' toString() by returning the height of this skyscraper

Marking notes

Marking categories

- Functional specifications
  - Does the program behave according to specifications?
  - Does it produce the correct output and pass all tests?
  - Are the class implemented properly?
  - Are you using appropriate data structures (if applicable)?

    - Does the code run properly on Gradescope (even if it runs on Eclipse, it is up to you to ensure it works on Gradescope to get the test marks)
- Non-functional specifications
    - Are there comments throughout the code (Javadocs or other comments)?
    - Are the variables and methods given appropriate, meaningful names?
    - Is the code clean and readable with proper indenting and white-space?
    - Is the code consistent regarding formatting and naming conventions?
- Penalties
    - Lateness: 10% per day
    - Submission error (i.e. missing files, too many files, etc.): 5%
    - "package" line at the top of a file: 5%
    - Compile or run-time error on Gradescope
    - Failure to follow instructions, (i.e. changing variable types, etc.)

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through cheating-detection software.

Submission (due Wednesday, June 23, 2021 at 11:55pm ET)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below. Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope and the test marks come directly from there. If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

ASSIGNMENT 1

Files to submit

- City.java
- Building.java
- Marina.java
- Skyscraper.java

Grading Criteria

- Total Marks: [20]
- Functional Specifications:
    - [2] City.java
    - [2] Building.java
    - [1] Marina.java
    - [1] Skyscraper.java
    - [10] Passing Tests
- Non-Functional Specifications:
    - [1.5] Meaningful variable names, private instance variables
    - [0.5] Code readability and indentation
    - [2] Code comments