CSC1130 Introduction to Computing Using Java 2021-2022 First Term

Department of Computer Science and Engineering
The Chinese University of Hong Kong

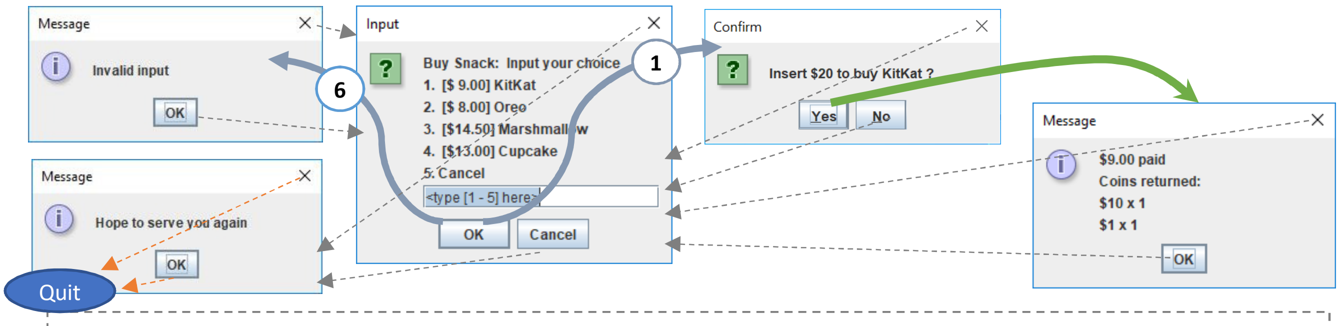
Assignment 3: **Snack Time**

Due date: 25 October 2021 (Mon) Full mark: 100
Expected normal time spent: 8 hours

Aim: 1. Practice defining classes, as well as creating and using objects.
2. Practice random number generation and more on structured programming.

Task: Create a Java application for a simple snack vending machine

The application starts with a Snack Time main menu. It allows the user to buy snacks using cash. A sample flow is shown below (in Windows, slight difference in other platforms):



Message
Invalid input
OK

Input
Buy Snack: Input your choice
1. [$ 9.00] KitKat
2. [$ 8.00] Oreo
3. [$14.50] Marshmallow
4. [$13.00] Cupcake
5. Cancel
<type [1 - 5] here>
OK Cancel

Confirm
Insert $20 to buy KitKat ?
Yes No

Message
$9.00 paid
Coins returned:
$10 x 1
$1 x 1
OK

Message
Hope to serve you again
OK

Snack Time properties:

In Snack Menu, user input is assumed to be **integer**, e.g., **1**, 2, 3, 4, 5, **6**, -9, 1000, etc.; **no type checking is needed**.

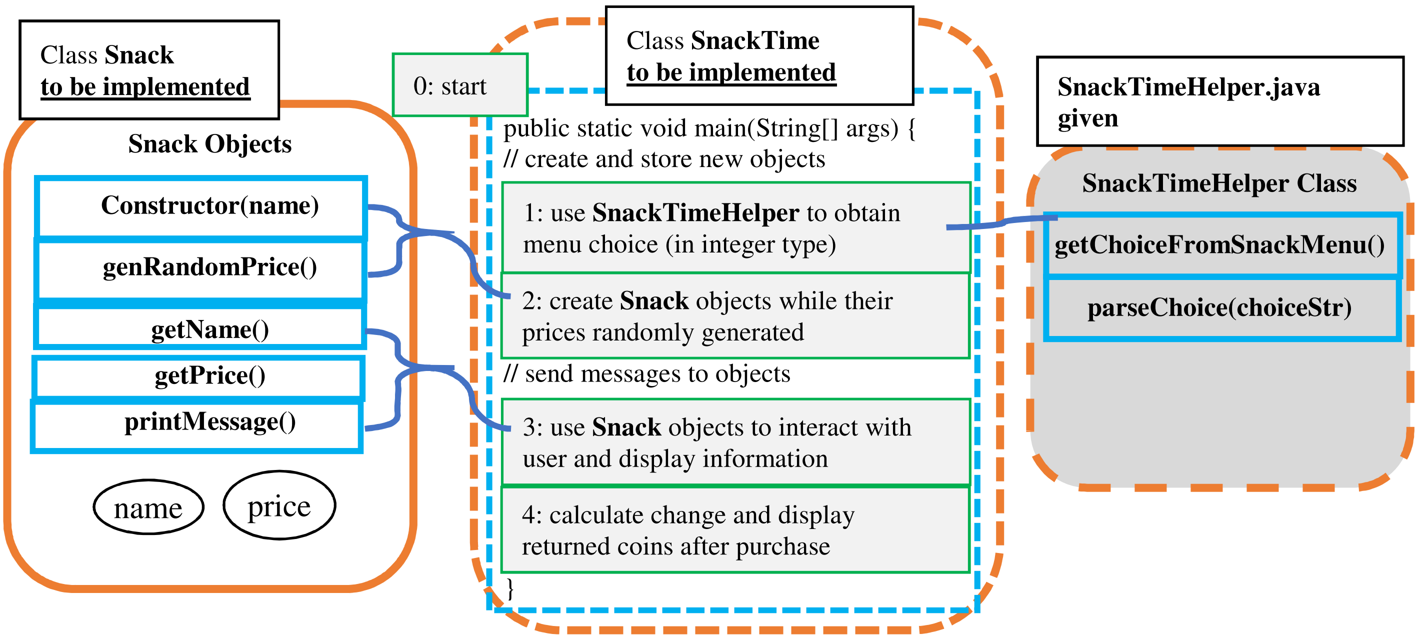
- If the user clicks `Cancel` or `Close` (×) dialogue buttons, pop up a Hope-dialog before quitting.
- If a choice input is ***out-of-range***, e.g., 6 , pop a warning dialog and then request to input again.

To buy a snack, e.g., 1 **for KitKat**, first confirm user to pay by cash of **$20**, that is **fixed and hard-coded**.

- If the user clicks `No` or `Close` (×) button, do nothing and return to the previous snack menu.
- If the user clicks `Yes` button, calculate the change, and pop another message dialog to show the returned coins (as a combination of $10, $5, $2, $1, 50c); when user clicks `OK`, finish buying, return to Snack Menu.

In general, if the user clicks `Close` (×) button in any message dialog, act as if the user had clicked `OK`.

Follow the flow of the whole application, explained in an OO diagram:


Class Snack
to be implemented
Snack Objects
Constructor(name)
genRandomPrice()
getName()
getPrice()
printMessage()
name
price
0: start
Class SnackTime
to be implemented
public static void main(String[] args) {
// create and store new objects
1: use SnackTimeHelper to obtain menu choice (in integer type)
2: create Snack objects while their prices randomly generated
// send messages to objects
3: use Snack objects to interact with user and display information
4: calculate change and display returned coins after purchase
}
SnackTimeHelper.java
given
SnackTimeHelper Class
getChoiceFromSnackMenu()
parseChoice(choiceStr)

Procedure (Step-by-Step):

1) Create a new NetBeans Java (with Ant) Application project `SnackTime`. Apply required package name `snacktime` and main class `SnackTime`. Create another class `Snack` under the package `snacktime`. You just have to complete these two classes. The source code for the class `SnackTimeHelper` is given for your use.

2) Define the `Snack` class with the following properties (you should figure out proper **modifiers**):

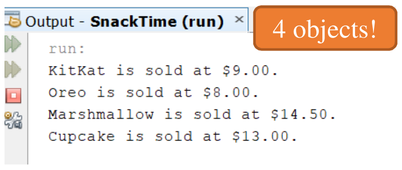
   a) Each `Snack` object, keeps several internal states in its fields, as shown previously.

      - The `name` of the snack, which is a *`String`*.
      - The `price` of the snack, which is a *`double`*.

   b) Define a **constructor** that receives parameter: `name` which is a *`String`*. This **constructor initializes** the fields and it **invokes method** `genRandomPrice()`.

   c) Define an instance method `getName()` which returns the `name` of the snack object in *`String`*.

   d) Define an instance method `getPrice()` which returns the `price` of the snack object, in *`double`*.



```
run:
KitKat is sold at $9.00.
Oreo is sold at $8.00.
Marshmallow is sold at $14.50.
Cupcake is sold at $13.00.
```

   e) Define an instance method `printMessage()` which **prints one line** of the `name` and the `price` of the snack object to the console, e.g., "KitKat is sold at $9.00" and returns nothing.

   f) Define a class method `genRandomPrice()` which returns a randomly generated price in range of [5.0, 15.0] of step size of $0.5. This method should be **used by the constructor** for randomly generating the price of each snack object on its creation.

3) Define the main class `SnackTime` according to the steps described in the previous OO diagram and flowchart. Define the main method, with other fields and methods, to perform JOptionPane dialogue user interactions as well as the **returning coins calculation** in the `SnackTime` class.

   a) Firstly, create four `Snack` objects. Snack names are given in the code fragment below. Prices are in 2 decimal places, the **price tags** of the snack should be **randomly generated during object construction**. Print the name and price of each snack to the console using the instance method `printMessage()` of each `Snack` objects, once each of them is created.

   b) Secondly, pop up a Snack Menu which requests user to input the choice of snack. Assume user always input integer value, no type checking in input is required.

      - On receiving a valid choice, proceed to the Payment Dialog which prompts user to pay by cash: only one $20 banknote is assumed (all snack prices are within $20), or
      - Pop up a warning dialog "Invalid input" for receiving an invalid choice from user; then return to the last input dialog.

   c) At the end of each successful purchase, calculate the changes by assuming available coins as **$10**, **$5**, **$2**, **$1**, **50c**. Show paid amount as well as the breakdown of the number of each coin to be returned.

d) Once the purchase is done, return to the Snack Menu to receive the next request from user.

4) Your class `SnackTime` should include at least the following fields and methods in addition to some other methods you define on your own:

```
    private static int[] coinsInCents;
    private static String[] snackNames;
/*** student's work to add extra fields here as needed ***/

    /**
     * Show a menu of choices and get user's input, given method
     * @return a String of user input: "1","2",…,"5", null means Cancel/Close
     */
    public static String showSnackMenu() {
/*** student's work to display snack menu items ***/
        return JOptionPane.showInputDialog("... " );
    }

/*** student's work to add extra methods here as needed ***/

    public static void main(String[] args) {
      // initialize coins, snack names & prices
      coinsInCents = new int[]{1000, 500, 200, 100, 50};
      snackNames = new String[]{"KitKat", "Oreo", "Marshmallow", "Cupcake"};

/*** student's work here to initialize values, call getChoiceFromSnackMenu()of
the SnackTimeHelper class, where Number-Format Exceptions are handled for you.
Modify the following sample code fragment to start processing user requests in a
loop ***/
        int snackMenuChoice = SnackTimeHelper.getChoiceFromSnackMenu();
        if (snackMenuChoice == -1)
            System.out.println("User closed or cancelled dialog box");
        else if (snackMenuChoice == 1)
            System.out.println("User picked 1");
        /* ... */
    }
```

5) You should implement the **`showSnackMenu()`** method to display all snack information and accept user input via `JOptionPane`.

6) Then, your **`main()`** method should repeatedly call **`getChoiceFromSnackMenu()`** from the class **`SnackTimeHelper`** to further process user requests obtained from **`SnackTime.showSnackMenu()`**.

7) Having obtained the user choice (in `int` type), you can proceed with the snack buying.

Submission:

1. Choose **Export Project** as **ZIP** via NetBeans menu, or

2. Locate your NetBeans project folder, e.g., `C:\Users\your_name\Documents\NetBeansProjects\SnackTime\`.

3. ZIP the whole NetBeans project folder **`SnackTime`**

4. Submit the file **`SnackTime.zip`** via our online Assignment Collection Box on Blackboard.

Marking Scheme and Notes:

1. Name your project, package and file correctly.

2. Include also your personal particulars and your academic honesty declaration statement in a header comment block of each source file.

3. The submitted program should be free of any typing mistakes, compilation errors and warnings.

4. Comment/remark, indentation, style is under assessment in every programming assignments unless specified otherwise. Variable naming, proper indentation for code blocks and adequate comments are important.

5. Remember to upload your submission and **click Submit** before 6:00 p.m. of the due date. No late submission would be accepted.

6. If you submit multiple times, ONLY the content and timestamp of the latest one would be counted. You may delete (i.e. take back) your attached file and re-submit. We ONLY take into account the last submission.

University Guideline for Plagiarism:

Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations.
Details may be found at http://www.cuhk.edu.hk/policy/academichonesty/.

With each assignment, students are required to submit a statement that they are aware of these policies, regulations, guidelines and procedures, in a header comment block.

Faculty of Engineering Guidelines to Academic Honesty:

MUST read: https://www.erg.cuhk.edu.hk/erg/AcademicHonesty
(you may need to access via CUHK campus network/ CUHK1x/ CUHK VPN)