420-201-VA | Assignment 1-A | Due by Friday, 31 January 2020

A Refresher Assignment on Programming 1 Topics

1 Purpose

The purpose of this assignment is to refresh your memory on object-oriented Java programming to the extent that it was covered in programming 1. To that end, this assignment will give you a sample object-oriented solution to an interactive game program, tasking you only with implementing (or coding) the given solution. The sample solution is designed to provide you with basic programming practice exercises, including the following:

- Write an interactive program
- Validate user inputs, avoiding mismatched and invalid values
- Use nested loops involving selections
- Generate random numbers within a specified range
- Keep track of running totals
- Practice breaking down a complex task into smaller and more manageable ones
- Write a pair of classes where one class is composed of ("has-a") a reference to the other
- Practice creating Java API pages for the classes we define
- Reuse code from our own packages of our own classes and methods
- Use simple arrays (which may be a new topic to some of you)

2 Slot Machines in a Nutshell

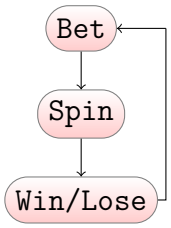
A slot machine is a classic coin-operated rip-off gambling machine. Traditional slot machines have three reels and one payline, with each reel adorned with about two dozen symbols. To use one, you insert coins into a slot and pull a handle that activates the spinning of the reels. After spinning a random number of times, the reels come to rest, showing three symbols lined up across the *payline*. If two or more of the symbols on the payline match, you will win a cash payout, which the slot machine dispenses back to you.

3 Assignment Overview

This assignment involves implementing a **SlotMachine** class whose objects can simulate the working of a simple slot machine.

Specifically, we are imagining and interested in a **SlotMachine** object that

- has 3 reels, each adorned with 7 fruit names as symbols
- interacts with the player, using the keyboard for input and the screen for output
- accepts the player's name at the time of its construction
- accepts the player's initial deposit (*of bet coins*) at the time of its construction
- can display an introduction to the game and its rules
- can repeat the following play cycle until the player chooses to stop:
  1. prompts the player for and read a bet value (*of bet coins*)
     - if the player bets more than its credits, then repeatedly prompts the player to enter a deposit value (of bet coins) until player's credits equals or exceeds the bet
  2. simulates spinning the reels,
  3. reports the spin outcome.



- keeps track of the following information:
  - player's name
  - total payouts (*of bet coins*) *won* by the user
  - total deposits (*of bet coins*) made by the player
  - total bets (*of bet coins*) placed by the player
  - current (most recent) bet value
  - total number of times the player has spun the reels

To facilitate understanding of the task at hand in this assignment, we are going to consider the expected behavior of **SlotMachine** objects from the perspective of client code.

3.1 Creating a slot machine

Here is how client code would like to create a slot machine object for a player named "Jack", setting its initial deposit to **10** bet coins.

```
1 public static void main(String[] args)
2 {
3    int initial_deposit = 10; // bet coins
4    Slot_Machine slot = new Slot_Machine("Jack", initial_deposit);
5
6    System.out.println(slot);
```

The **println** method call on line 6 will internally call the **toString()** method on **slot**, resulting in the following output.

```
1 -------------------
2 Slot Machine Status
```

```
3  ------------------
4  Player          : Jack
5  Current credits : 10
6  Total deposits  : 10
7  Total payouts   : 0
8  Total bets      : 0
9  Total spins     : 0
10 Bottom line     : No Loss, No Gain!
11 ------------------
```

As you can see in the output above, **SlotMachine**'s **toString()** override provides several pieces of information about the state of the **slot** object reference; specifically:

Line 4: Player's name

Line 5: Current credits (= total deposits + total payouts − total bets)

Line 6: total deposits (of bet coins) the player has made so far

Line 7: Total payouts (of bet coins) the player has won so far

Line 8: Total bets (of bet coins) the player has placed so far

Line 9: Total number of times the player has spun the reels

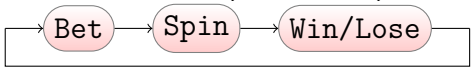
Line 10: Bottom line $= \begin{cases} \text{No Loss, No Gain} & \text{if } \text{diff} = \text{total payouts} - \text{total bets} = 0 \\ \text{Net Gain} = \text{diff} & \text{if diff} > 0 \\ \text{Net Loss} = -\text{diff} & \text{if diff} < 0 \end{cases}$

3.2 Starting the slot machine

Now, let's have our **slot** object reference play the game for the first time.

```
7     slot.play(); // jack starts playing the game
```

A sample output is shown in here, consisting of three parts:

- The initial part (lines 12-37) introduces the game and its rules. This part is displayed only once, specifically when the **slot** object reference *first* calls the **play()** method.
- The middle part (lines 38-40) is where the game proper is played, repeating the play cycle 

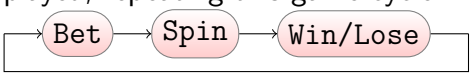
until the player chooses to stop the cycle.

  As you can see, the player chooses to quit the game, entering a 0 (zero) when prompted for a bet on line 40.
- The final part is the slot's final status report on lines 41-54.

```
12 Greetings Jack
13 Welcome to 3-Reel Slot Machine Game!
14 Each reel is adorned with the following 7 friut names:
15 Orange, Cherry, Lime, Apple, Banana, Peach, Melon
16
17 There are four possible types of payout combinations:
18 -----------------------------------------------------
19 1) Triple       : all 3 symbols match
20 2) Left-Double  : the left symbol matches either of the other two symbols
21 3) Right-Double : the center and rightmost symbols match
22 4) Zilch        : no matches
23
24 The Rules:
25 ----------
26 1) You will be prompted to enter a bet value.
27    A bet value is the number of bet coins you want to bet.
28    If your bet value exceeds your current balance, then
29    you'll have to deposit enough bet coins to satisfy your bet.
30 2) Enter 0 for a bet to end the game.
31 3) Get a Triple to win 3 times your bet.
32 4) Get a Left-Double to win 2 times your bet.
33 5) Get a Right-Double to win 1 time your bet.
34 6) Get a Zilch to lose your bet.
35
36 Let the Fun Begin!
37 Good Luck!
```

```
38
39 Credits: 10 bet coins    ← starts a game cycle
40 How many coins do you want to bet (0 to quit)? 0
```

This is where the game proper is played, repeating this game cycle:




```
41 -------------------
42 Slot Machine Status
43 -------------------
44 Player          : Jack
45 Current credits : 10
46 Total deposits  : 10
47 Total payouts   : 0
48 Total bets      : 0
49 Total spins     : 0
50 Bottom line     : No Loss, No Gain!
51 -------------------
52 Jack, your current game status is saved.
53 You may continue playing this game later.
54 Have a nice break and hurry back!
```

Entering 0 for a bet on line 40 causes this final part to be displayed before returning control back to client code where **play()** was called on **slot**.

As discussed above, lines 41-51 represent the string returned from **slot.toString()**.

Lines 52-54 are displayed to entice the player back to the game!

3.3 Actually playing the slot machine

Let's now have our **slot** object reference resume and actually play the slot machine:

```
8     // Jack stopped playing (1st break!)
9     // Jack resumes playing
10    slot.play();
```

Here is a sample output, which is also referenced here:

```
55
56  Welcome back Jack
57
58  Credits: 10 bet coins   ← starts a game cycle
59  How many coins do you want to bet (0 to quit)? 3
60
61  Here is your lucky spin of the reels ...
62
63  | Mandarin  |  Apricot  |  Apricot  |
64  |  Banana   |   Melon   |  Apricot  |
65  | Tangerine |    Fig    | Mandarin  |
66  |   Melon   |  Banana   |  Apricot  |
67  |   Pear    |    Fig    | Mandarin  |
68  |   Melon   |    Fig    | Tangerine |
69  ************************************
70  |   Pear    | Tangerine |    Fig    | ← payline
71  ************************************
72  You got Zilch - you win 0 bet coins
73
74  Credits: 7 bet coins   ← starts a game cycle
75  How many coins do you want to bet (0 to quit)? 5
76
77  Here is your lucky spin of the reels ...
78
79  | Tangerine |   Melon   | Mandarin  |
80  ************************************
81  |   Pear    |  Banana   |   Pear    | ← payline
82  ************************************
83  You got Left-Double - you win 10 bet coins
84
85  Credits: 12 bet coins   ← starts a game cycle
86  How many coins do you want to bet (0 to quit)? 0
```

Notice that on this second try, the **slot** object reference doesn't display the introduction part; instead, it welcomes the player by name on line 56.

Also notice that at the start of each spin, **slot** displays the current credits and then prompts the player to enter a bet value. (lines 58-59, 74-75, and 85-86)

To simulate spinning the reels, **slot** displays a *random* number of lines, each filled with three random fruit names, with the last line representing the *payline*.

To visually make the simulation of spinning the reels a bit more interesting, **slot** adorns the paylines with asterisks or stars (lines 69-71).

Below the paylines, on lines 72 and 83, **slot** displays the outcome of the spin.

```
87 ------------------
88 Slot Machine Status
89 ------------------
90 Player          : Jack
91 Current credits : 12
92 Total deposits  : 10
93 Total payouts   : 10
94 Total bets      : 8
95 Total spins     : 2
96 Bottom line     : Net Gain = 2
97 ------------------
98 Jack, your current game status is saved.
99 You may continue playing this game later.
100 Have a nice break and hurry back!
```

Once again, the player enters a 0 (zero) for a bet (on line 86). Hence, the final status report on lines 87-100.

Note that on line 96 the bottom line is reported as **Net Gain**, because total payouts exceeds total bets by 2.

3.4 Betting more than the credits

```
11    // Jack stopped playing (2nd break!)
12    // Jack resumes playing
13    slot.play();
```

As you can see on lines 105-113 in the output below, when the player enters a bet that exceeds the current credits, **slot** keeps prompting the player to deposit more bet coins.

←

```
101
102 Welcome back Jack
103
104 Credits: 12 bet coins   ← starts a game cycle
105 How many coins do you want to bet (0 to quit)? 17
106
107 Insufficient credits ->  (Credits = 12) but (Bet = 17)
108 To continue playing you must deposit at least 5 bet coins.
109 How many bet coins do you want to deposit (0 to quit)? 2
110
111 Insufficient credits ->  (Credits = 14) but (Bet = 17)
112 To continue playing you must deposit at least 3 bet coins.
113 How many bet coins do you want to deposit (0 to quit)? 4
114
115 Here is your lucky spin of the reels ...
116
117 | Tangerine |    Fig    |    Fig    |
118 | Mandarin  |   Melon   |   Melon   |
119 |   Melon   | Tangerine |   Pear    |
120 ************************************
```

```
121 |   Pear    |   Pear    |   Pear    |
122 ************************************
123 You got Triple - you win 51 bet coins
124
125 Credits: 52 bet coins   ← starts a game cycle
126 How many coins do you want to bet (0 to quit)? 0
```

Once again, the player enters a 0 (zero) for a bet (on line 126). Hence, the final status report on lines 127-140 below:

← →

```
127 -------------------
128 Slot Machine Status
129 -------------------
130 Player          : Jack
131 Current credits : 52
132 Total deposits  : 16
133 Total payouts   : 61
134 Total bets      : 25
135 Total spins     : 3
136 Bottom line     : Net Gain = 36
137 -------------------
138 Jack, your current game status is saved.
139 You may continue playing this game later.
140 Have a nice break and hurry back!
```

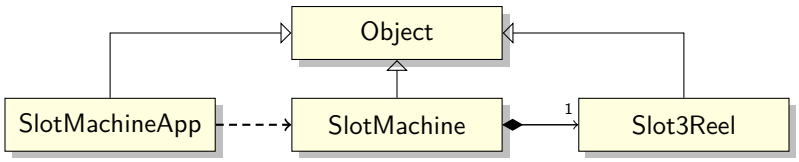
4 Assignment Requirements

This section assumes that you fully understand the task at hand in this assignment. Please let me know if you have any questions; in the meanwhile, please review section 3.

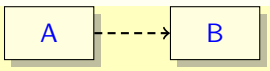
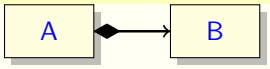
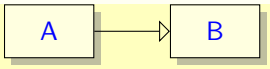
There are several ways to model a 3-reel slot machine. In this assignment, we will separate the play process from the spin process, arguing as follows:

- Delegate the task of representing and spinning reels to a **Slot3Reel** class. This will allow us to later on improve and enhance the class without affecting client code; for example, we may want replace it by a GUI version. Note that this class has no idea about the context in which it is used, and as such it has no idea about how the three random symbols it generates and maintains are interpreted in client code (for example, the **SlotMachine** class in this assignment).
- Delegate the task of playing the game and interacting with the player to a **SlotMachine** class, such that **SlotMachine** "has-a" **Slot3Reel** as an instance member, which in turn is responsible the slot machine's spin and payline simulation. Encapsulating the entire spin process in another class will allow **SlotMachine** to replace the spin process by another (say, GUI) version, without affecting the way it plays the game.

These ideas may also be expressed in UML notation as follows.



where the relationships among classes are defined as follows:

Relationship	UML notation	Meaning
Dependency		An **A** has a method that has a reference to **B**, as either a local variable or a parameter or return type.
Composition		An **A** is made up of **B**s with lifetime dependency: if the **A** is destroyed, its **B**s are destroyed as well.
Inheritance		An **A** is a special kind of **B**. **A** specializes **B**. **B** generalizes **A**.

4.1 Class Slot3Reel

Class **Slot3Reel** models a set of three slot reels, each adorned with 7 fruit name symbols:

```
Melon  Tangerine  Apricot  Fig  Mandarin  Pear  Banana
```

Class **Slot3Reel** should have the following members:

4.1.1 Private Attributes (Data)

symbolList A constant array of the seven symbols listed above.

payline An array of three symbols

4.1.2 Public Operations (Behavior)

Default constructor

Initializes **playline** with three randomly selected symbols.

String get(int k)

Returns the symbol at index k in **payline**. If **k** is negative or exceeds payline's length, the method throws an exception object of **IllegalArgumentException** passing to it the error message "**Slot3Reel**:get: array index out of bounds".

@Override String toString()

Forms and returns a string that contains the three symbols sandwiched and centered between four equally spaced vertical bars. For example,

```
|  Banana  |  Melon  |  Apricot  |
```

where the symbol represents a space.

@Override boolean equals(Object obj)

Determines whether the objects referenced by **this** and **obj** are of the same type and state. Consider the state of one **Slot2Reel** object equal to another, if their paylines each contain the same symbols at the same positions.

void spin()

Simulates spinning the three reels.

Ideally, the slots in a slot machine should each turn a random number of times so that they don't all stop spinning at the same time, creating a momentary suspense before revealing the spin outcome to the player.

In this first refresher assignment, we are going to have each slot turn exactly the same random number of times, say, $n, 2 \leq n \leq 9$. Implementing the following algorithm, we simply print **payline** n lines, refilling it with random symbols before each print. Note that the top $n - 1$ lines, printed inside the loop (steps 2-5), are

just for show. It is in Step 6, immediately after the loop, where the actual **payline** is defined.

Algorithm 1: Simulate spinning the reels

Input : None

Output:

- Text representation of spinning 3 slot reels on the screen.
- **payline**, whose three elements will determine the outcome of the spin

1 Generate a random integer, say n between 2 and 9, inclusive
2 **for** $k \leftarrow 1$ ***to*** $n - 1$ **do**
3 Fill **payline** with three randomly selected symbols
4 Print **payline**, formatted as shown on lines 63-68 on page 5
5 **end for**
6 Fill **payline** again on this final spin with three randomly selected symbols
7 Print **payline** within a pair of starred lines, jazzing it up visually, as shown on lines 69-71 on page 5
8 Return

4.2 Class SlotMachine

Class **SlotMachine** models a simple a 3-reel slot machine with the following characteristics.

4.2.1 Private Attributes (Data)

- a **reel** object reference to a **Slot3Reel** object representing the three reels of this slot machine
- player's name
- current bet value, the most recent bet placed by the player
- total bet values
- total deposits
- total payouts (wins)
- total number of spins

4.2.2 Public Operations (Behavior, Exposed to the outside world)

These public methods are available to all **SlotMachine** objects.

Normal constructor

Takes two parameters; initializes *this* player's name and initial deposit to the values passed in as parameters, in that order; sets the instance object reference **reel** to a **Slot3Reel** object and accepts the default values for all the other instance variables.

void play()

Algorithm 2: Plays a slot-machine game

Input : Non-negative Integer values entered on the keyboard.
Output: Textual simulation of a 3-reel slot machine on the screen.

```
2  if the calling object is a first time caller then
3      Introduce the game and rules to the player
4  else
5      Greet the player by name
6  Prompt the player for and read a non-negative bet value
7  while player's bet value ≠ 0 do
8      while the bet value exceeds the credits do
9          Prompt the player for and read a non-negative deposit amount
10         Increment total deposits by the value entered
11     Increment total bets by the bet value
12     Spin the reels calling reel's spin()
13     Compute payout using reel's payline
14     Increment total payouts by the spon's payout
15     Tell player about the spin outcome: "Triple", "Right-Double".
           "Left-Double". or "Zilch"
16     Prompt the player for and read a non-negative bet value
17 Display a final game report similar to 127-137.
18 Return
```

4.2.3 Private Operations (Behavior, Hidden from the outside world)

Performing single and focused task and typically taking a few lines of code to implement, these private methods facilitate implementation of other methods, and as such they are called *facilitator* methods. These methods (like back-office employees of a company) are responsible for providing internal operational support for other member methods. They are hidden from the outside world in the sense that they are not available to **SlotMachine** objects instantiated outside **SlotMachine**.

void intro()
Displays an introduction to the game and its rules exactly as shown on lines (lines 12-37) on page 4.

void finalGreetingMessage()
Displays final game status as shown on lines 41-54 on page 4.

void readBet()
Prompts the player for and reads a non-negative bet value (≥ 0) and then returns that value. For example see lines 38-40.

int readDeposit()
Called only if the player enters a bet that exceeds the current credits, this method

prompts the player for and reads a non-negative deposit (≥ 0) and then returns that value. For example see lines 107-109 and lines 111-113.

int credit()

Returns total deposits + total payouts − total bets

void validateBet()

Determines whether the current bet is negative or exceeds **credit()**

@Override public String toString()

Returns a string formatted as in lines 127-137.

public boolean isTriple()

Determines whether the 3 symbols on the payline are the same.

public boolean isLeftDouble()

Determines whether the left symbol on the payline is the same as either the middle or right symbol.

public boolean isRightDouble()

Determines whether the middle and right symbols on the payline are the same.

public boolean isZilch()

Determines whether no two symbols on the payline are the same.

public String getSpinOutcome()

Returns **"Triple"** for a triple, **"Left-Double"** for a left-double, **"Right-Double"** for a right-double; otherwise, returns **"Zilch"**.

int computeSpinPayout()

Algorithm 3: Compute Payout of a Spin

Input : The current bet and the **reel** control reference, both instance data members

Output: The spin's payout.

1 **if** **reel**'s *payline holds a triple* **then** payout $\leftarrow 3\times$ bet value
2 **else if** **reel**'s *payline holds a left-double* **then** payout $\leftarrow 2\times$ bet value
3 **else if** **reel**'s *payline holds a* right-double **then** payout $\leftarrow 1\times$ bet value
4 **else** payout $\leftarrow 0$
5 Return payout

5 Evaluation Criteria

Evaluation Criteria

Functionality	Ability to perform as required, producing correct output for any set of input data, Proper implementation of all specified requirements, Efficiency	60%
Robustness	Ability to handle input data of wrong type or invalid value	10%
OOP style	Encapsulating only the necessary data inside objects, Information hiding, Proper use of Java constructs and facilities.	10%
Documentation	Description of purpose of program, Javadoc comment style for all methods and fields, comments on non-trivial steps in all methods	10%
Presentation	Format, clarity, completeness of output, user friendly interface	5%
Code readability	Meaningful identifiers, indentation, spacing, localizing variables	5%