

To Be Submitted by Midnight, Nov. 18 (Monday)

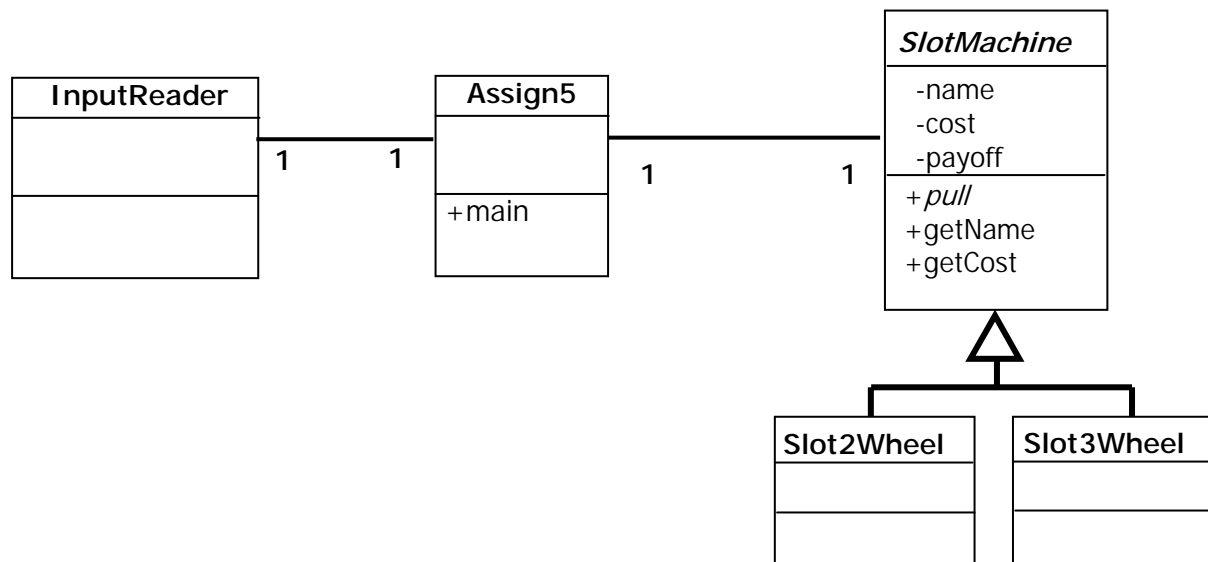
35 Points

New Concepts: Class inheritance, superclasses and subclasses, creating a new class from an existing class, abstract classes, using the keyword *super*, understanding polymorphism, casting objects and the *instanceof* operator, interfaces, arrays (declaring, creating, initializing and processing them), sorting algorithms, the bubble sort algorithm

PROBLEM STATEMENT:

In the last programming assignment, we broke the bank of the Low Rollers Casino. This time, we will expand on the Slot Machine idea but use a more realistic model, in terms of the slot machine operation and the cost to payoff averages.

In this assignment, you will use class inheritance to define two different types of slot machines. The overall design is shown below in a UML diagram:



Note that the **SlotMachine class is an abstract class**, and its **pull() method is an abstract method**. You will define two subclasses, the **Slot2Wheel** and the **Slot3Wheel** classes, as extensions of the abstract class. Some of the attributes and methods for the abstract class, SlotMachine, are shown in the UML diagram above. You may add more if desired, and you can decide which attributes and methods should be included in the other classes. *The only class that will contain a main method is the Assign5 class. All other classes must have constructors.* The **InputReader** class is our old favorite from assignments #3 and #4 and will be provided as before. As in the previous assignments, the name of the input file will be provided as a command line argument.

Here’s how the slot machines work. First, there are 5 types of fruit as defined below with their respective indices:

Index	1	2	3	4	5
Fruit	Cherry	Orange	Apple	Banana	Plum

For each “pull” of the slot machine, a fruit will be selected randomly for each wheel. Thus, for the Slot2Wheel class, you will need to generate 2 random numbers between 1 and 5 to represent the fruit selections. If the first random number is 1 and the second random number is 4, the pull just selected a

Cherry-Banana combination. For the Slot3Wheel class, you will generate 3 random numbers with each pull, to select a 3-fruit combination. For example, 1-4-1 is equal to Cherry-Banana-Cherry. As before, use the **java.util.Random** class for generating random numbers, starting with a seed of 1.

The payoffs will be determined according to the table below:

Payoff level	Combination 1 One cherry on any wheel	Combination 2 2 matches of any type fruit	Combination 3 3 matches of any type fruit
Payoff level 1	\$2	\$3	\$5
Payoff level 2	\$3	\$4	\$10
Payoff level 3	\$4	\$5	\$15
Payoff level 4	\$5	\$8	\$25

Each slot machine will have a designated payoff level that will be used to determine the payoff. In our example above, suppose the payoff level was 2; the Cherry-Banana combination (combination 1) pays off \$3 for level 2. If the combination selected randomly was 2 Plums (combination 2), the payoff for level 2 would be \$4. The net dollars made is the payoff minus the cost to play. (The net dollars might be negative.) NOTE: If a 3-wheel slot machine pulls a cherry and 2 matching fruit, use combination 2 (2 matches) to determine the payoff.

The **Assign5** class must contain the main method. This class will use the InputReader class to read lines of input from the input file. As before, each line of input describes one slot machine. For each line of input, your program will create a new SlotMachine object, **pull()** the handle 50 times, record the output of each **pull()**, and determine how much money overall would have been won or lost.

After evaluating all of the slot machines, your program must sort the slot machines twice and output a sorted list each time. The first sort is based on the total money generated by each slot machine; the second sort is an alphabetical ordering by slot machine name. In both lists, the name and the corresponding total money should be included. *Note that the sort must be done using a bubble sort method that you have written yourself;* the bubble sort algorithm will be discussed in the labs.

At the end of the program, an output is also generated to list the name of the machine that provided the most money (or lost the smallest amount of money).

THE INPUT FILE:

The first line in the input file will contain the number of possible slot machines (i.e., the number of lines contained in the rest of the input file). Each subsequent line contains the slot machine name (you may assume it will be one word long with no spaces), the number of wheels (2 or 3), the cost to play the machine, and the payoff level. For example, take the input line:

LuckyDucky 2 5 3

“LuckyDucky” is the name of the slot machine. It is a 2-wheel slot machine with a \$5 cost per pull, and a payoff level of 3.

SAMPLE INPUT:

```
7
LuckyDucky 2 5 3
Dubya Bush 4 3
Dubya_Bush 4 3 3
Dubya_Bush 3.3 4 5
SpruceGoose 2 20 5
SpruceGoose 2 20 4
Bubba_Gump 3 10 1
```

SAMPLE OUTPUT: (output numbers not necessarily correct)

LuckyDucky (2 wheels, \$5 per pull, level 3) Wins a total of \$50
Invalid input line ignored: Dubya Bush 4 3
Invalid input line ignored: Dubya_Bush 4 3 3
Invalid input line ignored: Dubya_Bush 3.3 4 5
Invalid input line ignored: SpruceGoose 2 20 5
SpruceGoose (2 wheels, \$20 per pull, level 4) Loses a total of \$25
Bubba_Gump (3 wheels, \$10 per pull, level 1) Wins a total of \$3

SORT BY PAYOFF
\$50 LuckyDucky
\$3 Bubba_Gump
\$-25 SpruceGoose

SORT BY NAME
Bubba_Gump \$3
LuckyDucky \$50
SpruceGoose \$-25

*** Results ***
Evaluated 3 slot machines
Most profitable: LuckyDucky, \$50

ERROR CHECKING:

Your program should include an error check of the *number of inputs on a line* and whether the inputs are *valid* and in the proper *order*. All valid inputs besides the slot machine names should be integers. The number of wheels per slot machine can only be 2 or 3. *Valid payoff levels are 1 through 4 only.* Invalid input lines should be ignored and an error message should be displayed with the invalid input line, such as the examples above.

INPUTS FOR PROGRAM SUBMISSION:

As in Assignments #3 and #4, the actual input file used during program submission will not be provided to students, but rather will be read automatically during the submission process. You should again make up your own input file to test your program. Include a variety of possible test cases to ensure that your program will handle any error included in the actual input file.

PROGRAM HEADER:

Include the following information in the program header, and fill in the information on the definition of the problem, the main algorithm steps, the expected inputs, the generated outputs, and the error checking performed

/******

NAME: XXXXXXXXXX XXXXXXXXXXXX ASSIGNMENT # 5
STUDENT# XXXXXX
SECTION# x
LAB TA: XXXXXXXXXX

DEFINITION OF PROBLEM:

MAIN ALGORITHM STEPS:

INPUT:

OUTPUT:

ERROR CHECKING:

*****/

GRADING:

- 11 points – Documentation (complete program header, informative identifier names, appropriate indentation and embedded comments)
 - 11 points – Programming style (consistent with standard java conventions, consistent with requirements given, corresponds to algorithm steps provided in the header)
 - 13 points – Output (correct values in the specified format, including error handling)
-

35 total points possible

COMPILER ERROR(S): 13 points deducted from your score (in lieu of output points)

RUNTIME ERROR: 7 points deducted from your score

GETTING HELP:

Students are required to correct most syntax errors and to bring a hardcopy of their program listing (complete with indicated errors and any output) with them when they wish to get help from the instructor or a Teaching Assistant (TA) or a Peer Learning Assistant (PLA) during office hours.

TA and PLA office hours are posted on the course web page:

<http://www.cecs.missouri.edu/~skubic/103>.

TAs, PLAs, and the instructor will assist students in understanding concepts, in finding elusive syntax errors, and in aiding correction of logic errors. However, they will NOT write or rewrite program code for students.