CSE1341 - Lab 5 Programming Assignment
Game Simulation with Multiple Methods
Assignment Due Saturday, October 27th, 2018 at 6:00 AM

This assignment will require a Pre-Lab and a Lab.

Our fifth Lab will be a baseball game simulation. You will build this in five incremental versions. Its best to build one increment successfully, then use that as the starting point for the next version.

If you are unfamiliar with the game of baseball, you will definitely learn the basics by completing this exercise!

Pre-Lab (5 POINTS): Create the BatterUp1 class with method headers for main and bat. Read the BatterUp1 instructions thoroughly and add pseudocode as comments within the body of each of the methods.

BatterUp1 [10 points]

Create a class called BatterUp1 which contains two methods: *main* and *bat*.

bat method
Simulate two die rolls by generating two random numbers in the range of 1 to 6 for each die. Store the die values in int variables and use these values to determine the result of batting. Print the two roll values, then at the end of the method, return an int which depicts a value based on the table below:

Die 1	Die 2	Represents	Return value
1	1	Single	1
2	2	Double	2
3	3	Triple	3
4	4	Home Run	4
Any other combination:			
Sum is even		Strike	-1
Sum is odd		Ball	-2

main method
Create a counter controlled repletion loop that calls the *bat* method ten times. Using the returned value, print the results of that swing of the bat.

BatterUp1 SAMPLE OUTPUT:

```
> java BatterUp1
  Rolled 3 4  BALL!
  Rolled 3 6  BALL!
  Rolled 5 5  STRIKE!
  Rolled 2 6  STRIKE!
  Rolled 3 6  BALL!
  Rolled 5 2  BALL!
  Rolled 4 3  BALL!
  Rolled 2 3  BALL!
  Rolled 3 2  BALL!
  Rolled 4 6  STRIKE!
```

BatterUp2 [20 points]

Make a copy of BatterUp1 and name it BatterUp2 which contains two methods: *main* and *bat*. Create a third method called *batterTakeTurn* which includes two int values initialized to 0 which will count the balls and strikes. The player should continue to bat (calling the *bat* method) until the player has had four balls, three strikes, or has had a hit (single, double, triple or home run.) Exit the loop when one of those conditions occurs.

Remove the previous code from the main method and simply call the *batterTakeTurn* method.

main method
Call *batterTakeTurn* once

bat method
Same as previous version

batterTakeTurn
Within a loop, continue to call the *bat* method until 3 strikes, 4 balls, or a hit (single, double, triple, or home run.) Add print statements to indicate the result of each bat. Hint: either use a boolean value or the break statement to exit the loop when one of these situations happens.

BatterUp2 SAMPLE OUTPUT:

```
> java BatterUp2
  Rolled 6 1  BALL!
  Rolled 6 4  STRIKE!
  Rolled 1 5  STRIKE!
  Rolled 1 5  STRIKE!
  Strike out!!
> java BatterUp2
  Rolled 2 5  BALL!
  Rolled 3 6  BALL!
  Rolled 3 3  Triple!
> java BatterUp2
  Rolled 5 2  BALL!
  Rolled 2 4  STRIKE!
  Rolled 4 4  Home Run!
```

BatterUp3 [20 points]

Make a copy of BatterUp2 and name it BatterUp3 which contains two methods: *main*, *bat* and *batterTakeTurn*.

Add a static String array in the class to contain nine player names. Pick any nine names to populate the player names.

Add a return type of boolean (eg isOut) to batterTakeTurn, which returns true if the player's turn is over due to a strike out., otherwise it returns false. In main, use this returned boolean value to increment the outs counter and the next player is up. Continue to loop through the batting order repeatedly (starting over at the beginning of the array after the last batter). When three outs have occurred, break out of that loop and end.

Change the *main* method to loop through all nine players continuously until three outs have occurred. In this game, the only way to get an out is by striking out.

Additional changes to the main method:

- Declare an int variable to count the outs. Initialize to 0
- Loop through the players array, for each player
    - Print out the name of player at bat along with the total outs
    - Change that player's location from -1 (dugout) to 0 (at bat)
    - Player continues to bat until the batterTakeTurn method returns true (Strike out)
- Wrap that *for* loop for all players in a *while* loop that continues until 3 outs
- if all batters have batted without three outs, the while loop will cause the batting order to repeat until three outs have occurred

BatterUp3 SAMPLE OUTPUT:

```
> java BatterUp3
Amy is at bat with 0 outs
  Rolled 4 2  STRIKE!
  Rolled 3 5  STRIKE!
  Rolled 2 6  STRIKE!
  Strike out!!
Bob is at bat with 1 outs
  Rolled 3 3  Triple!
Carl is at bat with 1 outs
  Rolled 5 4  BALL!
  Rolled 5 3  STRIKE!
  Rolled 4 1  BALL!
  Rolled 3 2  BALL!
  Rolled 6 5  BALL!
  Walk
Diana is at bat with 1 outs
  Rolled 1 1  Single!
Ed is at bat with 1 outs
  Rolled 5 5  STRIKE!
  Rolled 1 4  BALL!
  Rolled 3 6  BALL!
  Rolled 3 2  BALL!
  Rolled 5 5  STRIKE!
  Rolled 6 6  STRIKE!
  Strike out!!
Francis is at bat with 2 outs
  Rolled 4 1  BALL!
  Rolled 6 6  STRIKE!
  Rolled 3 1  STRIKE!
  Rolled 4 1  BALL!
  Rolled 1 3  STRIKE!
  Strike out!!
3 outs - next team!
```

BatterUp4 [30 points]

Make a copy of BatterUp3 and name it BatterUp4 which contains four methods: : *main*, *bat, batterTakeTurn*, and ***movePlayers****.*

Make the following changes

Outside the methods, inside the class:

- Add another static array in the class with player locations.
- Initialize the array contents to all -1 values Note: Player location can have a value of 0 (at bat) 1 (first base) 2 (second base) 3 (third base) 4+ (home base) or -1 (dugout)
- Add an int variable to record the score. Initialize to zero

In the main method:

- At the beginning of a player's turn before batting, change the location of that player (in the locations array) from -1 (dugout) to 0 (at bat)
- If the batterTakeTurn method returns true, the player has struck out so that player's location should be set to -1 (return to dugout)

Add a new method named movePlayers, with an int parameter:

- The *int* value passed in indicates the number of bases each player on base should move ahead (any location with any value other than -1)
- After any hit or walk, all players on base will advance the same number of (Walk=1, Single=1, Double=2, Triple=3, and Home Run = 4) Note that some players may have a value greater than 4 at this point.
    - Use a loop to go through the locations array and add the value from the parameter to each of the players that don't have a value of -1
- Add another loop which loops through the locations array and increments the score for each player with a value of 4 or more. After updating the score for a player, that players location should be set to -1 (return to dugout.)
    - If a player scores, print the message "<player name> scored!!"

Change the batterTakeTurn method:

- Within the *if* statements in *batterTakeTurn,* call the *movePlayers* method if the player got a single, double, triple, home run or walk.
- Add a print statement before each player starts a turn which shows the current score.

BatterUp4 SAMPLE OUTPUT:

```
> java BatterUp4

SCORE: 0
Amy is at bat with 0 outs
  Rolled 5 2  BALL!
  Rolled 6 1  BALL!
  Rolled 2 5  BALL!
  Rolled 6 6  STRIKE!
  Rolled 3 1  STRIKE!
  Rolled 6 3  BALL!
  Walk

SCORE: 0
Bob is at bat with 0 outs
  Rolled 4 2  STRIKE!
  Rolled 1 2  BALL!
  Rolled 3 4  BALL!
  Rolled 2 4  STRIKE!
  Rolled 1 1  Single!
```

```
SCORE: 0
Carl is at bat with 0 outs
  Rolled 1 5  STRIKE!
  Rolled 3 5  STRIKE!
  Rolled 2 6  STRIKE!
  Strike out!!

SCORE: 0
Diana is at bat with 1 outs
  Rolled 6 6  STRIKE!
  Rolled 3 2  BALL!
  Rolled 2 6  STRIKE!
  Rolled 2 3  BALL!
  Rolled 6 2  STRIKE!
  Strike out!!

SCORE: 0
Ed is at bat with 2 outs
  Rolled 1 4  BALL!
  Rolled 2 5  BALL!
  Rolled 2 4  STRIKE!
  Rolled 2 3  BALL!
  Rolled 6 3  BALL!
  Walk

SCORE: 0
Francis is at bat with 2 outs
  Rolled 4 4  Home Run!
Amy scored!
Bob scored!
Ed scored!
Francis scored!

SCORE: 4
Georgia is at bat with 2 outs
  Rolled 6 1  BALL!
  Rolled 6 1  BALL!
  Rolled 3 4  BALL!
  Rolled 3 4  BALL!
  Walk

SCORE: 4
Hank is at bat with 2 outs
  Rolled 1 5  STRIKE!
  Rolled 1 1  Single!

SCORE: 4
Izzy is at bat with 2 outs
  Rolled 1 4  BALL!
  Rolled 1 5  STRIKE!
  Rolled 2 4  STRIKE!
  Rolled 2 6  STRIKE!
  Strike out!!
3 outs - next team!
```

BatterUp5 [15 points]

Make a copy of BatterUp4 and name it BatterUp5 which contains five methods: : *main*, *bat, batterTakeTurn*, *movePlayers,* and ***displayField****.*

Make
Add new method named displayField:

- This method will which print the three bases in the field with the name of the player currently on that base. If there is no player on that base, print "empty"
    - Hint: create three String variables that contain the value "empty", then loop through the locations array and replace its value with the player name (from the players array) if their location is 1, 2 or 3

Change the main method:

- Call the displayField method at the beginning of a player's turn after printing the score.

BatterUp5 SAMPLE OUTPUT:

```
>java BatterUp5

SCORE: 0
[ 1 ] empty  [ 2 ] empty  [ 3 ] empty
Amy is at bat with 0 outs
  Rolled 1 2  BALL!
  Rolled 2 4  STRIKE!
  Rolled 3 6  BALL!
  Rolled 5 1  STRIKE!
  Rolled 4 6  STRIKE!
  Strike out!!

SCORE: 0
[ 1 ] empty  [ 2 ] empty  [ 3 ] empty
Bob is at bat with 1 outs
  Rolled 1 5  STRIKE!
  Rolled 4 3  BALL!
  Rolled 6 5  BALL!
  Rolled 1 4  BALL!
  Rolled 5 2  BALL!
  Walk

SCORE: 0
[ 1 ] Bob  [ 2 ] empty  [ 3 ] empty
Carl is at bat with 1 outs
  Rolled 1 4  BALL!
  Rolled 6 3  BALL!
  Rolled 4 1  BALL!
  Rolled 4 5  BALL!
  Walk

SCORE: 0
[ 1 ] Carl  [ 2 ] Bob  [ 3 ] empty
```

```
Diana is at bat with 1 outs
  Rolled 4 6  STRIKE!
  Rolled 2 3  BALL!
  Rolled 3 3  Triple!
Bob scored!
Carl scored!

SCORE: 2
[ 1 ] empty  [ 2 ] empty  [ 3 ] Diana
Ed is at bat with 1 outs
  Rolled 6 2  STRIKE!
  Rolled 3 2  BALL!
  Rolled 1 5  STRIKE!
  Rolled 5 2  BALL!
  Rolled 1 4  BALL!
  Rolled 1 5  STRIKE!
  Strike out!!

SCORE: 2
[ 1 ] empty  [ 2 ] empty  [ 3 ] Diana
Francis is at bat with 2 outs
  Rolled 1 4  BALL!
  Rolled 1 5  STRIKE!
  Rolled 1 5  STRIKE!
  Rolled 2 5  BALL!
  Rolled 6 6  STRIKE!
  Strike out!!
3 outs - next team!
```

NOTES: Comment your program to explain your steps. Each program should compile without errors and should run to produce outputs described for each exercise. The following points will be discounted if the related element is missing or incorrect:

- Output formatting monetary amounts (dollar sign and 2 places after the decimal point) [2 points each]
- Proper names for classes, variables, and methods [1 point each]
- No Comments [5 points]
- Program doesn't compile [5 points for each minor error up to 5 errors provided that after fixing the errors the program compiles. If the program does not compiler after the 5 errors are fixed, partial credit will be given not to exceed 50 points]
- Source code (java file) missing [15 points]
- Executable (class file) missing [15 points]
- Both java file and class file missing [100 points]
- Missing method where required [5 points each]