CSE1341 - Lab 6 Assignment

Overview

In Lab 5 you created the BatterUp baseball simulation using structured programming techniques. In this lab, you will rewrite your application using object-oriented programming.

Pre-Lab (5 Points)

Create the class named *Base* found in the instructions. Include the *name* attribute, constructor and getter and setter methods for *name*.

Lab (95 Points)

Create the BatterUp system using object oriented programming, following the design provided on the following pages. Your output should match the format shown on the last page, although your actual output will vary based on the the outcome of the game.

Submit the *java* and *class* files via Canvas (as a single zip-file). Include a comment block at the top of each *Java* file that includes your name, student id number, and "Lab 6-Fall 2018". Also be sure to include the answers to the post-lab questions found in these instructions.

NOTES:

Each program should include comments that explain what each block of code is doing. Additionally, the programs should compile without errors, and run with the results described in the exercise. The following deductions will be made from each exercise if any of the following is incorrect or missing:

Proper formatting [5 points]
Proper names for classes and variables [5 points]
Comments [5 points per class]
Program doesn't compile [10 points]
Source code (java file) missing [10 points]
Executable (class file) missing [10 points]
Missing array where an array was required [5 points each]
Missing loop where a loop was required [5 points each]
Missing class from the design provided [10 points each]
Missing method from the design provided [5 points each]

This Lab is due Saturday November 3 at 6:00am.

Launcher

+main(String[]):void

1 → 1

BatterUp

-outs : int
-score: int
-nextPlayerIndex: int

+BatterUp()
+createPlayers():void
+getNextPlayer():Player
+play():void
+movePlayers(int):void
+displayField():void

1 → 1 theField

Field

+Field()
+getDugout() : Base
+getBatterBox() : Base
+moveAhead(Base, int) : Base

1 → 6 bases

1 → 9 players

Player

-name: String
-strikes: int
-balls : int

+Player(String, Base)
+getName() : String
+setName(String) : void
+getLocation() : Base
+setLocation(Base) : void
+isNotInDugout() : boolean
+takeTurn() : int
+bat() : int
+toString() : String

1 → 1 location

Base

-name: String

+Base(String)
+getName() : String
+setName(String) : void
+isDugout() : boolean
+isHome() : boolean
+toString(): String

Base methods:

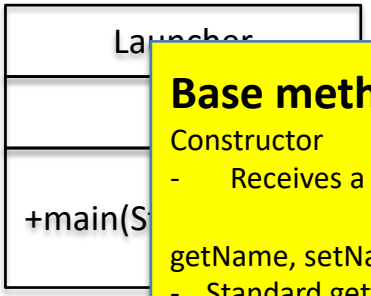
Constructor
- Receives a String as a parameter. Use this to set the value of the name attribute.

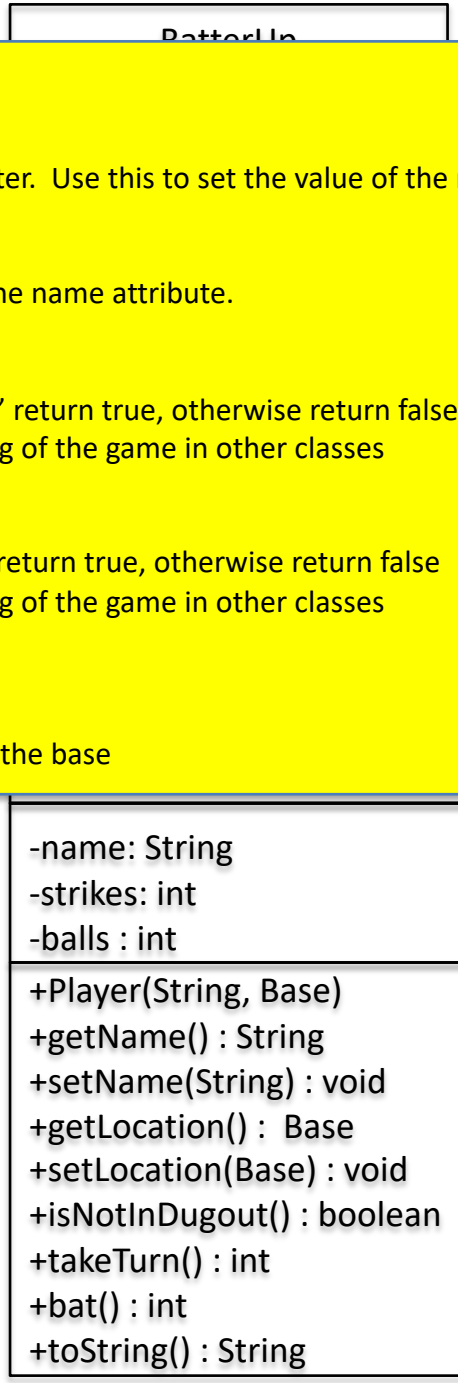
getName, setName:
- Standard getter and setter for the name attribute.

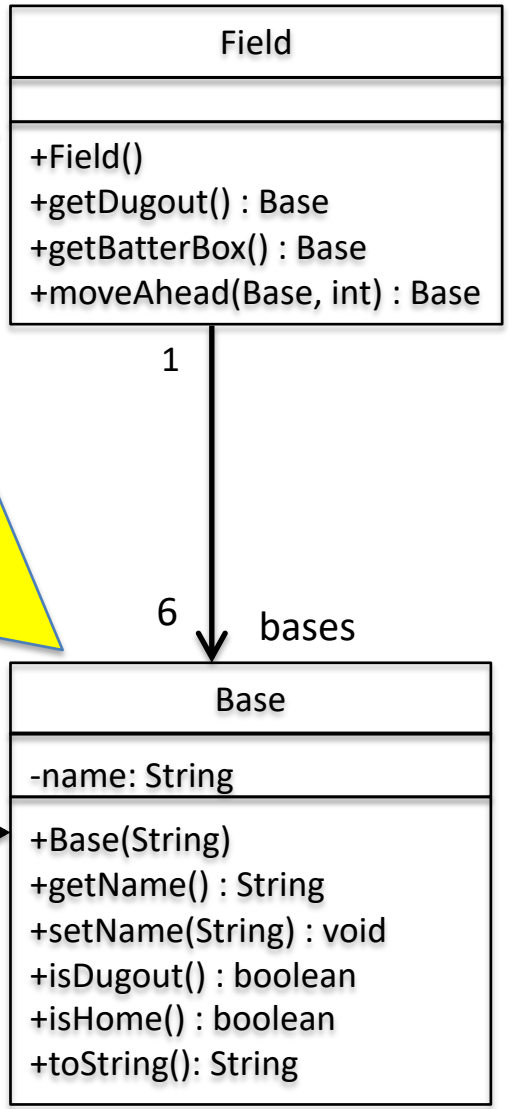
isDugout:
- If the value of name is “Dugout” return true, otherwise return false
- This is needed during the playing of the game in other classes

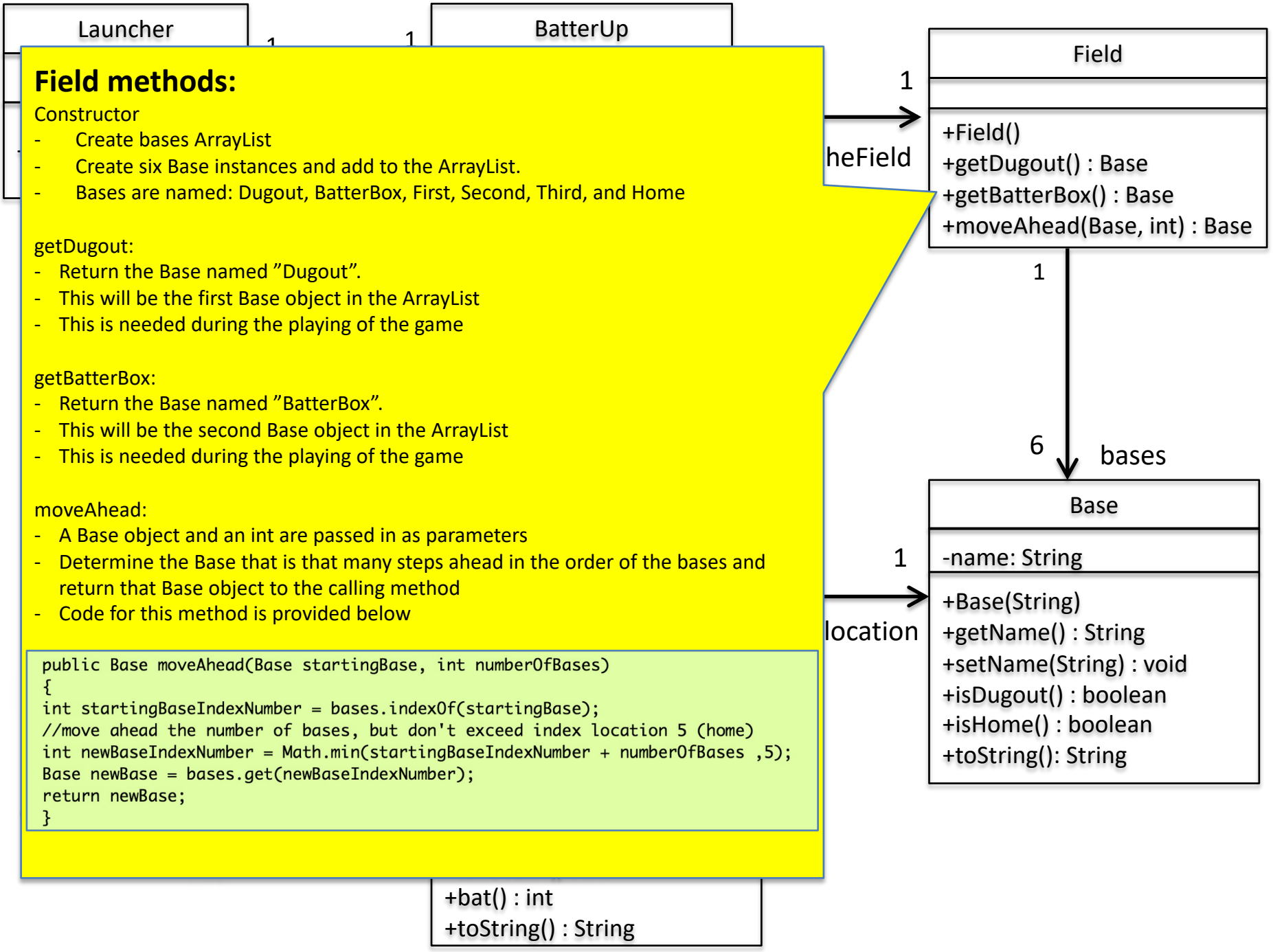
isHome:
- If the value of name is “Home” return true, otherwise return false
- This is needed during the playing of the game in other classes

toString
- Return a string representation of the base









Field methods:

Constructor
- Create bases ArrayList
- Create six Base instances and add to the ArrayList.
- Bases are named: Dugout, BatterBox, First, Second, Third, and Home

getDugout:
- Return the Base named ”Dugout”.
- This will be the first Base object in the ArrayList
- This is needed during the playing of the game

getBatterBox:
- Return the Base named ”BatterBox”.
- This will be the second Base object in the ArrayList
- This is needed during the playing of the game

moveAhead:
- A Base object and an int are passed in as parameters
- Determine the Base that is that many steps ahead in the order of the bases and return that Base object to the calling method
- Code for this method is provided below

```
public Base moveAhead(Base startingBase, int numberOfBases)
{
int startingBaseIndexNumber = bases.indexOf(startingBase);
//move ahead the number of bases, but don't exceed index location 5 (home)
int newBaseIndexNumber = Math.min(startingBaseIndexNumber + numberOfBases ,5);
Base newBase = bases.get(newBaseIndexNumber);
return newBase;
}
```

Player methods:

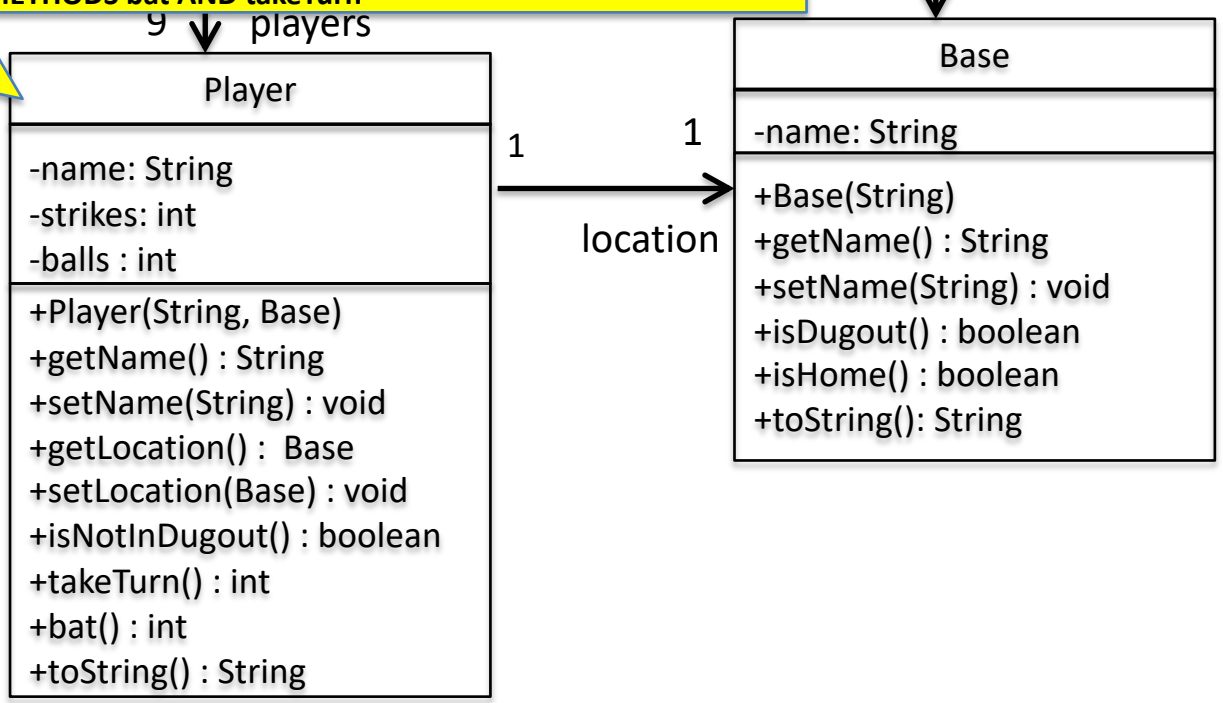
Constructor
- Receives a String and a Base instance as parameters
- Use the String to set the value of name, and assign the Base object to the location attribute
- (This Base should be the Dugout, which is each player's starting location when the game starts

getName, setName, getLocation, setLocation:
- Standard getter and setter for the name and location attributes.

isNotInDugout:
- Call the isDugout method in the Base that is the player's location, which will return a boolean
- Return true if the player is not in the Dugout, otherwise return false
- This is used during the playing of the game

toString
- Return a string representation of the Player

SEE THE NEXT PAGE FOR NOTES ABOUT THE METHODS bat AND takeTurn



Player
-name: String -strikes: int -balls : int
+Player(String, Base) +getName() : String +setName(String) : void +getLocation() : Base +setLocation(Base) : void +isNotInDugout() : boolean +takeTurn() : int +bat() : int +toString() : String

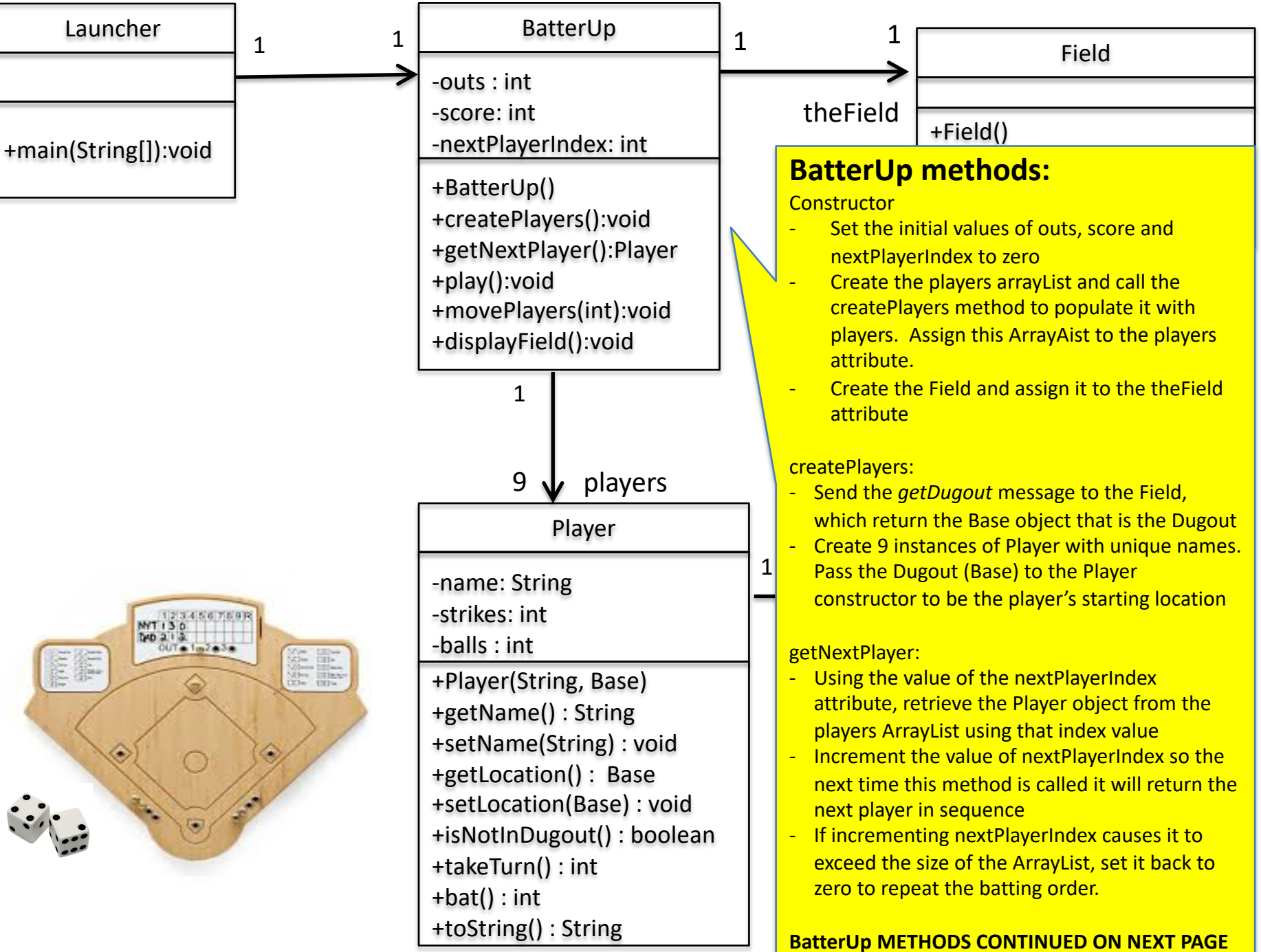
Player methods: (continued)

bat

- Simulate rolling two 6-sided dice and compare the values
- Using the rules of the game from the previous lab, determine whether the player hit the ball, got a ball, or got a strike.
- Use that value to calculate the total number of bases the player can move forward after this bat. (e.g., Single – 1, Double – 2, Triple – 3, Home Run – 4, Ball or Strike - 0.
- If the player got a ball, increment the balls counter
- If the player got a strike, increment the strikes counter
- Return that value to the calling method for use there

takeTurn

- This method contains the logic for one player's turn at bat, which always ends with a hit, a walk, or a strike out
- First, set the values of strikes and balls attributes to zero to start the turn at bat
- Next, create a loop which continues until the player gets a hit, four balls, or three strikes.
- Call the *bat* method (described above) and use the value it returns (0..4) to determine whether or not the player got a hit
    - If > 0 break out of the loop and return the number of bases to move (1-4)
    - If 0, check to see if the player struck out or needs to walk
        - If 4 balls, break out of the loop end return the value 1 to the calling method because the player can now walk, which is moving 1 base ahead
        - If 3 strikes, break out of the loop and end the turn by returning 0 to the calling method in the BatterUp class.



BatterUp methods:

Constructor

- Set the initial values of outs, score and nextPlayerIndex to zero
- Create the players arrayList and call the createPlayers method to populate it with players. Assign this ArrayAist to the players attribute.
- Create the Field and assign it to the theField attribute

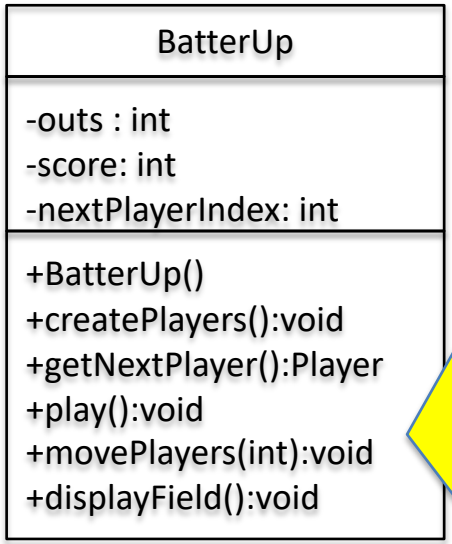
createPlayers:

- Send the *getDugout* message to the Field, which return the Base object that is the Dugout
- Create 9 instances of Player with unique names. Pass the Dugout (Base) to the Player constructor to be the player's starting location

getNextPlayer:

- Using the value of the nextPlayerIndex attribute, retrieve the Player object from the players ArrayList using that index value
- Increment the value of nextPlayerIndex so the next time this method is called it will return the next player in sequence
- If incrementing nextPlayerIndex causes it to exceed the size of the ArrayList, set it back to zero to repeat the batting order.

BatterUp METHODS CONTINUED ON NEXT PAGE



BatterUp methods: (Continued)

play

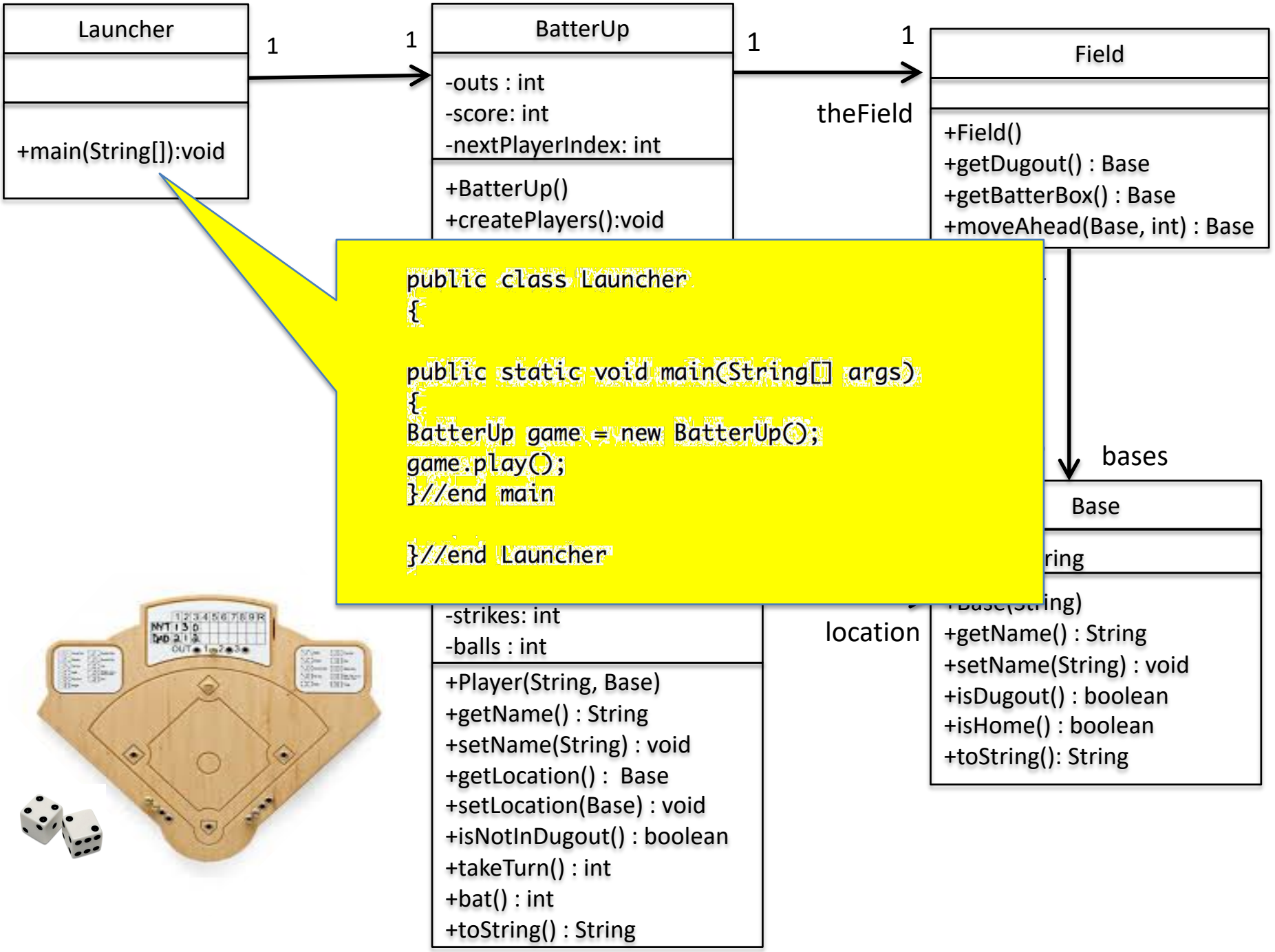
- Create a loop that continues until the value of the outs attribute is 3, which ends the game.
- Within the loop:
  - Call displayField
  - Call getNextPlayer and store that Player in a temporary variable to be used in this method
  - Call the player's getName method to print the message "<*name*> is batting"
  - Call the field's getBatterBox method to get the Base that is the BatterBox
  - Set the current player's location to that Base (BatterBox)
  - Send the takeTurn message to the current player, which will return a value of 0, 1, 2, 3 or 4
    - If 0, increment the outs counter and set the player's location to the Dugout (using the Field's getDugout method)
    - If 1-4, call the MovePlayers method, passing that number as a parameter
  - Exit the loop when the outs counter reaches 3. Print a final message with the score

movePlayers

- Receive an int parameter which has the number of bases to move every player forward
- Create a loop that loops through the entire players ArrayList
- Retrieve each player, one at a time
  - Send the isNotInDugout message to the Player
  - If false, do nothing and continue the loop with the next player
  - If true, call the Field's *moveAhead* method, passing the value passed in as a parameter. It will return a Base object which is the player's new location
  - Using Player's setLocation method, set the Player location to this new Base object
  - Using the isHome method in Base, determine if the Player is now on Home Base
    - If true, increment the score and print a message that the player scored
    - After incrementing the score, set the player's location back to the Dugout (using Field's getDugout method)
  - Continue the loop unilt all player's have been moved or skipped

displayField

- Print the occupants of first, second and third base using the same logic you created in Lab 5


Launcher
+main(String[]):void
1
1
BatterUp
-outs : int
-score: int
-nextPlayerIndex: int
+BatterUp()
+createPlayers():void
1
1
theField
Field
+Field()
+getDugout() : Base
+getBatterBox() : Base
+moveAhead(Base, int) : Base
bases
Base
+getName() : String
+setName(String) : void
+isDugout() : boolean
+isHome() : boolean
+toString(): String
location
-strikes: int
-balls : int
+Player(String, Base)
+getName() : String
+setName(String) : void
+getLocation() : Base
+setLocation(Base) : void
+isNotInDugout() : boolean
+takeTurn() : int
+bat() : int
+toString() : String
public class Launcher
{
public static void main(String[] args)
{
BatterUp game = new BatterUp();
game.play();
}//end main
}//end Launcher

Sample Output:

```
$ java Launcher

SCORE: 0

[ 1 ] empty  [ 2 ] empty  [ 3 ] empty

Amy is batting
  Rolled 6 3  BALL!
  Rolled 4 5  BALL!
  Rolled 3 4  BALL!
  Rolled 2 4  STRIKE!
  Rolled 4 1  BALL!
  Walk

SCORE: 0

[ 1 ] Amy  [ 2 ] empty  [ 3 ] empty


Bob is batting
  Rolled 3 5  STRIKE!
  Rolled 1 3  STRIKE!
  Rolled 1 5  STRIKE!
  Strike out!!

SCORE: 0

[ 1 ] Amy  [ 2 ] empty  [ 3 ] empty

Carl is batting
  Rolled 2 3  BALL!
  Rolled 5 2  BALL!
  Rolled 1 2  BALL!
  Rolled 6 4  STRIKE!
  Rolled 3 6  BALL!
  Walk

SCORE: 0


[ 1 ] Carl  [ 2 ] Amy  [ 3 ] empty


Diana is batting
  Rolled 1 3  STRIKE!
  Rolled 5 5  STRIKE!
  Rolled 4 1  BALL!
  Rolled 1 5  STRIKE!
  Strike out!!
```

```
SCORE: 0


[ 1 ] Carl  [ 2 ] Amy  [ 3 ] empty


Ed is batting
  Rolled 1 2  BALL!
  Rolled 1 4  BALL!
  Rolled 3 4  BALL!
  Rolled 4 1  BALL!
  Walk


SCORE: 0


[ 1 ] Ed  [ 2 ] Carl  [ 3 ] Amy


Francis is batting
  Rolled 6 4  STRIKE!
  Rolled 5 5  STRIKE!
  Rolled 2 2  Double!
Amy SCORED!!
Carl SCORED!!


SCORE: 2


[ 1 ] empty  [ 2 ] Francis  [ 3 ] Ed


Georgia is batting
  Rolled 1 6  BALL!
  Rolled 4 4  Home Run!
Ed SCORED!!
Francis SCORED!!
Georgia SCORED!!
```

```
SCORE: 5


[ 1 ] empty  [ 2 ] empty  [ 3 ] empty


Hank is batting
  Rolled 6 5  BALL!
  Rolled 5 5  STRIKE!
  Rolled 4 2  STRIKE!
  Rolled 2 2  Double!


SCORE: 5


[ 1 ] empty  [ 2 ] Hank  [ 3 ] empty


Izzy is batting
  Rolled 3 1  STRIKE!
  Rolled 3 1  STRIKE!
  Rolled 2 3  BALL!
  Rolled 6 6  STRIKE!
  Strike out!!

THREE OUTS!
GAME OVER WITH A SCORE OF 5
```