

Homework 6: Baseball

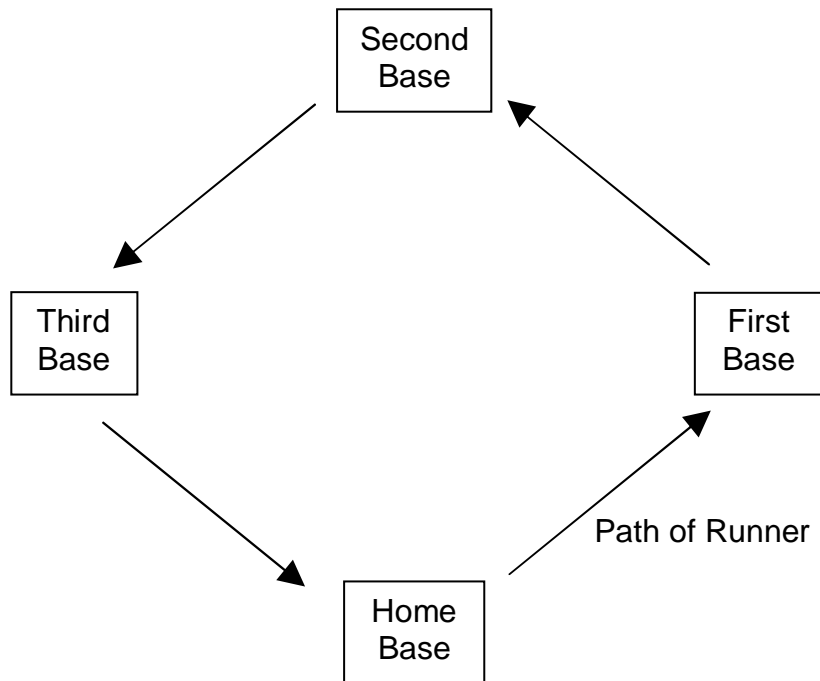
Due: March 1 at 11:55pm

This program should be completed using pair programming. (See Pair Programming Guidelines in Moodle.)

In this assignment, you will write a Java class to simulate part of a game of baseball, using simplified rules.

Simplified Rules for Baseball

Baseball is played on a field containing four *bases*, called First Base, Second Base, Third Base, and Home Base. At any given time, each of First Base, Second Base, and Third Base may be unoccupied (empty) or may be occupied by a player called a *runner*. Home Base is never occupied by a runner. The runners are all trying to advance through the sequence of bases (First, Second, Third, Home), as shown in the figure below. A runner who arrives at Home Base scores a *run*. After scoring a run, the runner is no longer a runner and no longer occupies a base. The objective for a baseball team is to score as many runs as possible.



A baseball game consists of several *innings*. An inning lasts until three players become *out*. During the inning, a series of players have turns as *batter*, in which they stand at Home Base and attempt to hit a thrown baseball with a bat. There are four possible outcomes when a player takes a turn as batter:

1. A *strikeout*: The batter becomes out. If this is the third out, the inning is over. No runners advance.

2. A *base hit*: A base hit is worth N bases where $1 \leq N \leq 4$. The batter becomes a runner and advances N bases in the base sequence (example: in a 3-base hit, the batter becomes a runner and advances to Third Base.) In addition, all runners who currently occupy bases also advance N bases in the base sequence (example: in a 2-base hit, a runner on First Base advances to Third Base.) Any runner (including the batter) who advances to Home Base scores a run.
3. A *sacrifice*: The batter becomes out and does not become a runner. All the runners who are already on a base advance one base through the base sequence. For example, a runner on First Base advances to Second Base. If, as the result of a sacrifice, a runner advances to Home Base, he scores a run (even if the batter becomes the third out of the inning.) If this is the third out, the inning is over.
4. A *walk*: The batter advances to First Base and becomes a runner. If there is already a runner on First Base, he advances to Second Base. If a runner advances to Second Base but there is already a runner on Second Base, the runner on Second Base advances to Third Base. If a runner advances to Third Base but there is already a runner on Third Base, the runner on Third Base advances to Home Base and scores a run.

You will write a Java class named `Inning`. The internal state of your `Inning` class should include everything you need to simulate a baseball inning (for example, the number of runs, the number of outs, and which bases are occupied by runners.) Your `Inning` class should have a zero-argument constructor that initializes the `Inning` to have no runs or outs and no bases occupied. The `Inning` class must also implement the following methods:

```
boolean strikeout()
boolean baseHit(int bases)
boolean sacrifice()
boolean walk()
void print()
```

The first four of these methods (`strikeout`, `baseHit`, `sacrifice`, and `walk`) update the internal state of the `Inning` class based on the outcome of a batter, as described by the method name and parameter, and return `true` if the inning is still in progress (less than three outs) or `false` if the inning is over (3 outs). The `print` method prints one line describing the number of runs, the number of outs, and the bases that are occupied by runners (for example, `1 2` indicates runners on First Base and Second Base.) The `print` method should use the format shown in the following examples:

```
runs = 2, outs = 2, occupied bases: 1 2
runs = 0, outs = 1, occupied bases: none
```

In addition to the `Inning` class described above, your program must contain a `Baseball` class that has a `main` method. The `main` method constructs an `Inning` object and then

reads a series of events from `System.in`. The events are described by single letters and numbers, separated by white space, as follows:

S indicates a strikeout.

H 3 indicates a base hit worth 3 bases (The digit may be from 1 to 4. Note the whitespace between the H and the digit.)

C indicates a sacrifice.

W indicates a walk.

The `main` method of `Baseball` should read the series of events from the `System.in`. After reading each event, `main` should do the following:

1. Print a description of the event, and the number of bases if it is a base hit. Examples: Strikeout, 2-Base Hit, Sacrifice, Walk.
2. Call the appropriate method of the `Inning` object to simulate the event.
3. Call the `print` method of the `Inning` object to print the state of the `Inning` after simulating the event.
4. Print a blank line to separate this event from the following event.

The `main` method should continue processing events in this way until the result of an `Inning` method indicates that the inning is over. At this point, the `main` method should print the line, "End of Inning". Any input remaining in `System.in` should be ignored (the input may contain unneeded events that are not part of the current inning, to test how your program decides when the inning is over.)

The following illustrates an example input and the expected output for this input. Your output should follow the format shown in this example.

Example Input:

```
W H 2 S C W H 1 H 3 H 4 S W H 4 S
```

Example Output:

```
Walk
runs = 0, outs = 0, occupied bases: 1

2-Base Hit
runs = 0, outs = 0, occupied bases: 2 3

Strikeout
runs = 0, outs = 1, occupied bases: 2 3

Sacrifice
runs = 1, outs = 2, occupied bases: 3

Walk
runs = 1, outs = 2, occupied bases: 1 3
```

```
1-Base Hit
runs = 2, outs = 2, occupied bases: 1 2

3-Base Hit
runs = 4, outs = 2, occupied bases: 3

4-Base Hit
runs = 6, outs = 2, occupied bases: none

Strikeout
runs = 6, outs = 3, occupied bases: none

End of Inning
```

Your program should not prompt for input. You should assume that the input will be redirected from an input file, and that the output will be saved in an output file. For example, your program might be invoked as follows:

```
java Baseball <myInput.txt >myOutput.txt
```

You should test your program using the sample input and output shown above. The sample input and output files can be downloaded from Moodle. The graders will test your program using a different input file.

Your program should be based on the simplified rules listed above. You do not need to simulate all the other rules of a real baseball game (double plays, stolen bases, the infield fly rule, etc.)

Submit one file named `Baseball.java` to Moodle before the assignment deadline. You do not need to submit the output of the program. Each partner should submit a file, as described in Pair Programming Guidelines.