SPR CONSULTING
POWERING BUSINESS WITH TECHNOLOGY

Software Requirements for HorseTrack

A programming assignment for SPR Consulting applicants

Problem Description

Your task is to develop a simulation for a automated teller machine for Horse Race Park.

The teller machine you develop only has two responsibilities:

1. To allow a park employee to set the number of the winning horse.
2. To allow patrons to determine if the horse they have already bet on won, and then collect their winnings if it did.

Upon startup, the machine should display a list of its current inventory of money, followed by a list of horses and current race results. At this point the program accepts user input.

The specified input and output formats for the machine must be followed exactly. At the end of these instructions, you will find examples of some input/output scenarios.

Definitions

- **Odds:** A measure of the likelihood that a horse will win the race, as made by race organizers. Odds represent the number of races, on average, a horse would have to enter and run in order to be expected to win one race. For example, The Darn Gray Cat has an *odds* value of 5, meaning their likelihood of winning this race is estimated to be 1 in 5. Note that the higher the odds value, the less likely the horse is thought to win the race – and therefore the higher the payout should they win.

Processing Requirements

Ticket winnings will be calculated based on the size of the bet, and the odds of the selected horse to have won the race. Payout is to be calculated as *bet amount * odds.*

The following table lists the available horses and their associated odds:

#	Horse Name	Odds
1	That Darn Gray Cat	5
2	Fort Utopia	10
3	Count Sheep	9
4	Ms Traitour	4
5	Real Princess	3
6	Pa Kettle	5
7	Gin Stinger	6

The machine must dispense a payout using the ***minimum number of bills possible***, constrained by its current inventory (cash on hand). If the machine cannot dispense a payout due to insufficent funds, an error message is displayed (see sample output below).

The following table lists the denominations (bills) the machine contains, and the maximum inventory for each denomination:

Denomination	Inventory
$1	10
$5	10
$10	10
$20	10
$100	10

At startup, the machine should be fully stocked with cash (each denomination should be assumed to be stocked to the maximum number of bills allowed). The machine can be restocked to its initial state at any time. Restocking the machine should restore each denomination to the initial inventory amount.

There is no limit on how much a patron can wager on a particular race, but all bets must be in increments of $1. For example, $1, $3, $10, $1000 are all valid bets, but $1.50 isn't a valid bet. You must display an error message if the user enters an invalid bet.

Input Format

Your solution should read from the standard input stream, one command per line. No prompts or other extraneous user messages should be displayed. Blank input lines should be ignored.

Valid commands are as follows:

- 'R' or 'r' - restocks the cash inventory
- 'Q' or 'q' - quits the application
- 'W' or 'w' [1-7] - sets the winning horse number
- [1-7] <amount> - specifies the horse wagered on and the amount of the bet

If the user enters an improperly formatted command, then the program should display a single-line message with the following format:

```
Invalid Command: <characters that were entered>
```

If the user tries to specify a non-existent horse number, then the program should display a single-line message with the following format:

```
Invalid Horse Number: <number that was entered>
```

If the user enters the number of a horse that did not win, the program should display a single-line message with the following format:

```
No Payout: <horse name>
```

If the user enters an invalid bet amount, the program should display a single-line message with the following format:

```
Invalid Bet: <amount>
```

If the user enters a horse number that did win, the program should display a multi-line message with the following format:

```
Payout: <horse name>,<total winnings>
Dispensing:
$1: <number of $1 bills>
$5: <number of $5 bills>
$10: <number of $10 bills>
$20: <number of $20 bills>
$100: <number of $100 bills>
```

However, if the machine does not have enough cash on hand to make a complete and exact payout, the program should display a single-line message with the following format:

```
Insufficient Funds: <payout amount>
```

The machine inventory and list of horses (see next section) should be displayed immediately following any applicable message.

At application startup horse number 1 is considered the winner. The winner can be changed using the 'w' command as outlined above.

Output Format

All output should be written to the standard output stream. At program startup, and following the processing of every command, the machine inventory and the horse list should be displayed. Both the inventory and horse lists should be ordered as shown above

```
Inventory:
<denomination>,<quantity in inventory>
 ...
<denomination>,<quantity in inventory>
Menu:
<horse number>,<horse name>,<odds>,<did-win>
 ...
<horse number>,<horse name>,<odds>,<did-win>
```

The `<did-win>` indicator should be either "won" or "lost".

Note: the sample output is indented in these instructions to make it easier to read. The output generated by your program should not have any whitespace at the beginning of a line.

Technical Notes:

Your solution should be a command-line program written in Java, C#, VB.NET, Ruby or Groovy. You may use a language not included in this list only if you receive prior permission from SPR to do so. If you use any external libraries in developing your solution (i.e. libraries that are not part of the standard Java or .NET platform) then you should bundle these libraries with your code so that we can run your solution.

It is not required that the initial machine configuration (cash inventory, list of horses, etc.) be dynamic. In particular, it is acceptable to perform this initialization in code, rather than reading the configuration from an external file or database. However, your program should be flexible enough to allow inventory or horses to be added or deleted without requiring extensive code changes.

The following qualities will be considered when evaluating your solution:

- Functionality (solution satisfies these requirements)
- Soundness and appropriateness of overall design
- Clarity, readability and maintainability of source code
- Quality of unit tests, if provided
- Packaging (ease with which we can open, install, and execute the solution)

Example:

Upon application startup, the initial inventory and horse lists would look like this:

```
Inventory:
$1,10
$5,10
```

```
$10,10
$20,10
$100,10
Horses:
1,That Darn Gray Cat,5,won
2,Fort Utopia,10,lost
3,Count Sheep,9,lost
4,Ms Traitour,4,lost
5,Real Princess,3,lost
6,Pa Kettle,5,lost
7,Gin Stinger,6,lost
```

For input consisting of the following commands:

```
1 55
2 25
W 4
4 10.25
```

the program would produce the following output (including the startup output):

```
Inventory:
$1,10
$5,10
$10,10
$20,10
$100,10
Horses:
1,That Darn Gray Cat,5,won
2,Fort Utopia,10,lost
3,Count Sheep,9,lost
4,Ms Traitour,4,lost
5,Real Princess,3,lost
6,Pa Kettle,5,lost
7,Gin Stinger,6,lost
Payout: That Darn Gray Cat,$275
Dispensing:
$1,0
$5,1
$10,1
$20,3
$100,2
Inventory:
$1,10
$5,9
$10,9
$20,7
$100,8
Horses:
1,That Darn Gray Cat,5,won
2,Fort Utopia,10,lost
3,Count Sheep,9,lost
4,Ms Traitour,4,lost
```

```
5,Real Princess,3,lost
6,Pa Kettle,5,lost
7,Gin Stinger,6,lost
No Payout: Fort Utopia
Inventory:
$1,10
$5,9
$10,9
$20,7
$100,8
Horses:
1,That Darn Gray Cat,5,won
2,Fort Utopia,10,lost
3,Count Sheep,9,lost
4,Ms Traitour,4,lost
5,Real Princess,3,lost
6,Pa Kettle,5,lost
7,Gin Stinger,6,lost
Inventory:
$1,10
$5,9
$10,9
$20,7
$100,8
Horses:
1,That Darn Gray Cat,5,lost
2,Fort Utopia,10,lost
3,Count Sheep,9,lost
4,Ms Traitour,4,won
5,Real Princess,3,lost
6,Pa Kettle,5,lost
7,Gin Stinger,6,lost
Invalid Bet: 10.25
```