










Specification

Be sure to read the entire specification carefully! In particular, see the Program Structure at the bottom of the page. A suggested Development Strategy is provided on the next slide.


Table of Contents

-  Program Behavior
-  User Input
 -  Checking for Valid User Input
-  Priority Score
 -  Score Calculation
 -  Hospital Zip Code
 -  Priority Determination
-  Daily Statistics
-  Program Structure

Program Behavior

You can see an example interaction with the program below (user input is **bold** and underlined):

The program begins by printing a short introduction. Then, the program prompts for a patient's name, followed by the patient information that will be used to compute the **priority score** (see below). The program uses this information to compute a score and determine the patient's priority. This information is printed out, along with a brief recommendation based on the

priority. The program then prints a thank you message and prompts for another patient name. This process continues until the word `quit` is typed as the name, at which point summary statistics for the day are printed (see the  Daily Statistics section below).

Note that your output will not have the underlined and bolded text like the above -- this is just for the purposes of the example!

User Input

You should use the `Scanner` class to read in all user input through the console. For numerical user input, you should use the `nextInt()` or `nextDouble()` method. For all other user input, you should use the `next()` method.

Be very careful **NOT** to use the `Scanner`'s `nextLine()` method! Some very unintuitive and tricky bugs arise when you mix `nextLine()` and `nextInt()` calls on the same `Scanner`, so make sure to only use `next()` for non-numerical user input in this assessment.

You may assume that all of the user's answers will be a **single token of the appropriate type**. In other words, where the program prompts for and expects an `int` input, you can assume that the user's answer will be an `int`. Similarly, where the program prompts for and expects a `String` response, you can assume that the user's answer will be a `String`, and where the program prompts for a `double`, the user will enter a `double`.

Checking for Valid User Input

You may assume that the user enters reasonable or "valid" responses when prompted for each piece of information, with two exceptions:

- **You should verify that the user enters a 5-digit integer when prompted for the patient's zip code.** The formula for calculating the patient score relies on the first and second digits of the zip code, so we need to make sure that the user enters a valid zip code to factor it into the formula appropriately.
- **You should verify that the user enters an integer in the range 1-10 (inclusive) when prompted for the patient's pain level.** The pain level of the patient directly affects the calculated priority score, so you should ensure the entered pain level is within the appropriate range.

If the user enters an invalid zip code, you should reprompt the user with the message `"Invalid zip code, enter valid zip code: "` *until* they enter a valid zip code. Similarly, if the user enters an invalid pain level, you should reprompt the user with the message `"Invalid pain level, enter valid pain level (1-10): "` *until* they enter a valid pain level.

For example:

The method to check that an integer has exactly 5 digits is provided for you (`fiveDigits`). You just need to call it appropriately in your code! Notice that the `fiveDigits` method returns a `boolean` - think carefully about how you should use this return type to ***reprompt the user until they enter a valid zip code.***

Priority Score

We use the information that the user inputs ("*features*") to calculate a **priority score** for the patient. The patient features used for the calculation are:

- age
- zip code
- whether this hospital is "in network"
 - an answer of *y* or *yes* indicates that the hospital is "in network" for the patient
 - *any other response* indicates that the hospital is not "in network" for the patient
- pain level
- temperature

Score Calculation

Each patient starts out with a base score of 100, which then is increased or decreased based on the inputted information as follows:

- If the age of the patient is a child (younger than 12 years old) or a senior citizen (at least 75 years old), add 50 to their score.
- If the *first digit* of their zip code is the same as the first digit of the *hospital's* zip code, add 25 to their score. If the *second digit* of their zip code **also** matches the second digit of the hospital's zip code, add another 15 to their score.
 - NOTE: If the first digit of each zip code doesn't match, then it doesn't matter whether the second digits of each zip code match - no change should be made to the patient's priority score.
- If the hospital is "in-network" for the patient's medical insurance, add 50 to their score.
- Take the patient's reported pain level, multiply it by 10, and add it to their score.
- If the patient's temperature is above 99.5 degrees Fahrenheit, add 8 to their score.

For a walkthrough of how Jun's, Emma's, and Johan's scores were calculated in the first sample execution above, click "Expand."

Expand

Jun

- Start with a score of 100.
- Jun's age is reported to be 5, so Jun is considered a child and we add 50 to their score: 150.
- Jun's zip code is reported to be 44467, so the first digit of their zip code doesn't match the hospital's zip code (12345) so no change is made.
- The hospital is "in network" for Jun, so we add 50 to their score: 200.
- Jun's pain level is 2, so we add $2 * 10$ or 20 to their score: 220.
- Jun's temperature is 99.8, which is above our threshold of 99.5 so we add 8 to their score: **228** which is their final priority score.

Emma

- Start with a score of 100.
- Emma's age is reported to be 77, so Emma is considered a senior citizen and we add 50 to their score: 150.
- Emma's zip code is 12487, so the first digit of their zip code matches the hospital's zip code (12345) and we add 25 to their score: 175. Then we check the second digit of Emma's zip code, which also matches the hospital's so we add another 15 to their score: 190.
- The hospital is "in network" for Emma, so we add 50 to their score: 240.
- Emma's pain level is 10, so we add $10 * 10$ or 100 to their score: 340.
- Emma's temperature is 101.7, which is above our threshold of 99.5 so we add 8 to their score: **348** which is their final priority score.

Johan

- Start with a score of 100.
- Johan's age is reported to be 22, so Johan is neither a child nor a senior citizen and no change is made.
- Johan's zip code is reported to be 92107, so the first digit of their zip code doesn't match the hospital's zip code (12345) so no change is made.
 - NOTE: Even though the second digits of Johan's and the hospital's zip codes match, we still don't make any change to their score because the first digits don't match. We only check the second digits if we already know that the first digits match!
- The hospital is not "in network" for Johan, so no change is made.
- Johan's pain level is 5, so we add $10 * 5$ or 50 to their score: 150.
- Johan's temperature is 104.3, which is above our threshold of 99.5 so we add 8 to their score: **158** which is their final priority score.



Hospital Zip Code

Your program should use a **class constant** to represent the hospital's zip code, named `HOSPITAL_ZIP`. The default hospital zip code will be `12345` (and this is what these sample logs use) but you should be able to change the value stored in this class constant to another 5-digit value and have the rest of the program operate seamlessly.

Make sure to correctly declare the class constant as `public`, `static`, and `final`!



Priority Determination

Once the priority score is determined, each patient is placed into one of three different priority groups based on their score.

- **High priority:** Patients with scores that are greater than or equal to 333 are considered "high priority", and the following output should be printed.

We have determined this patient is high priority, and it is advised to call an appropriate medical provider ASAP.

- **Medium priority:** Patients with scores greater than or equal to 168 but less than 333 are considered "medium priority", and the following output should be printed.

We have determined this patient is medium priority. Please assign an appropriate medical provider to their case and check back in with the patient's condition in a little while.

- **Low priority:** Patients with scores less than 168 are considered "low priority", and the following output should be printed.

We have determined this patient is low priority. Please put them on the waitlist for when a medical provider becomes available.



Daily Statistics

The program should continue to process patients until the user quits the program by entering `quit` (exactly like this, all lowercase) when asked for the next patient's name. Once the user does this, the program should output some statistics about the patients that it evaluated that day. In particular, it should print

- The total number of patients processed

- The priority score of the highest priority patient it processed

Program Structure

Your program should utilize methods, parameters, and returns to add a clear and understandable structure to the code. In particular, your code should include **at least the following 7 methods** with the indicated parameters and return values:

1. A method to print the program's introduction message
 - no parameters
 - no return value
2. A method to get a patient's name through user input
 - takes one `Scanner` parameter
 - returns the name read from input
3. A method to collect the required patient information (age, zip code, insurance information, pain level, and temperature) and compute the priority score
 - takes one `Scanner` parameter
 - returns the computed score
4. A method to calculate the priority score of the patient (**this method should be called in the above method**)
 - takes five parameters, once for each patient feature
 - returns the computed score
5. A method to print the patient's priority
 - takes two parameters-- the name and the score
 - no return value
6. A method to print out the overall statistics for the day
 - takes two parameters-- the number of patients and the maximum score
 - no return value
7. A method to find out if the zip code has 5 digits (**this is provided in your workspace**)

With the exception of methods 4 and 7, all these methods should be called from `main`. Method 4 should be called from method 3. Method 7 should be called where ever you check that the user input is valid.

You may include additional methods if you wish, but you **must** include these methods as defined. (Naturally, your program will also include a `main` method.)

Along with the required methods above, you **must have a loop in** `main` which allows the program to keep asking for new patient information until the user chooses to quit.