# Design

In the first block of code, we import the packages, which include the **Scanner** and **Random** classes that we will use later in the code. We also define the **main** method, which is the entry point of the program. Inside the **main** method, we create a **scanner** object to read input from the console and a **random** object to generate random numbers. We also create an **ArrayList** called **possibleNumbers** to store all the possible 6-digit combinations (0 to 999,999 inclusive) that the computer could guess. We use a loop to add all 6-digit combinations to the **possibleNumbers** list. Next, we generate a random number from the **possibleNumbers** list. The computer’s first guess will always be 123456. We start an infinite loop, which will run until the computer guesses the secret code correctly or the user provides inconsistent feedback. Inside the loop, we print the current guess and prompt the user to enter the number of digits that are correct and in the correct position, and the number of correct digits that are in the wrong position. We store the user's input in the variables **correctInPlace** and **correctOutOfPlace,** respectively. If **correctInPlace** is equal to 6, it means the computer has correctly guessed the secret code, so we print a message to the console and break out of the loop. Inside the loop, we create a new **ArrayList** called **newPossibleNumbers** to store the numbers that match the given feedback. We use a nested loop to iterate through each number in the **possibleNumbers** list, and for each number, we count the number of digits that are correct and in the correct position (inPlace) and the number of correct digits that are in the wrong position (outOfPlace). If the number of correct digits in the correct position and the number of correct digits in the wrong position for the current number match the values of **correctInPlace** and **correctOutOfPlace,** respectively, we add the current number to the **newPossibleNumbers** list. After we have iterated through all the numbers in the **possibleNumbers** list, we check the size of the **newPossibleNumbers** list. If it is 0, it means that there are no numbers in the **possibleNumbers** list that match the given feedback, which means the user has provided inconsistent feedback. In this case, we print a message to the console and break out of the loop. If the size of the **newPossibleNumbers** list is not 0, it means that there are one or more numbers in the **possibleNumbers** list that match the given feedback. In this case, we update the **possibleNumbers** list to be the **newPossibleNumbers** list and generate a new random guess from the updated possibleNumbers list. The loop then continues, and the computer will make a new guess and prompt the user for feedback. Finally, we have the **getRandomNumber** function, which takes a **Random** object and an **ArrayList** of possible numbers as input and returns a random number from the list. It does this by generating a random index between 0 and the size of the list (exclusive) and returning the element at that index.