Mastermind in Java

Mastermind was invented in 1970 by Mordecai Meirowitz, an Israeli postmaster and telecommunications expert, however the gameplay is centuries old.



Figure 1 A Mastermind set

Mastermind is a two-player game. One player (the computer in our case) is the codemaker; and the other is the codebreaker (the player).

The codemaker selects 4 coloured “code pegs” and hides them from the codebreaker. There are 6 colours to choose from and the codemaker may select the same colour code peg multiple times.

The codebreaker then has up to 12 guesses to work out the correct code (i.e. guess the right pegs in the right order). Each time the codebreaker guesses by placing 4 code pegs on the board, the codemaker responds with 1 to 4 *key pegs*. A black key peg indicates that the guess contains a colour in the *correct* place. A white peg indicates that the guess contains a colour contained in the code, but in the *wrong* place.

Once the codebreaker gets 4 black key pegs, they have guessed the code and won the game. If the codebreaker runs out of space on the board (after 12 guesses) the codemaker wins.

Only one key peg can be awarded per peg in the guess, for example, if the code was  (red, green, blue, blue) and the guess was  (blue, yellow, yellow, blue) then the result would be . I.e. one black peg for the last blue code peg and one white peg for the first blue code peg.

# Structure of the Code

The code structure is fairly straightforward with only 3 classes.

1. Mastermind – the main game in a class.
2. CodePeg – represents a coloured peg.
3. KeyPeg – represents a monochrome peg.

We’re going to use “enums” (short for enumerations) for the peg classes. These are special types of class that declare all the possible instances of that class. In the case of code pegs, this is one for each colour.

## Creating the Project & Enums

Create a new Java project File -> New -> Java Project and call it Mastermind. Remember, in Java, everything is case sensitive, so take care to get capital letters in the right place.

Right click the “src” package and select New -> Enum. Enter CodePeg as the name of the enum and click finish:

**public** **enum** CodePeg {

***RED***, ***GREEN***, ***BLUE***, ***YELLOW***, ***MAGENTA***, ***CYAN***;

**private** **char** firstChar;

**private** CodePeg() {

**this**.firstChar = **this**.name().charAt(0);

}

**public** **boolean** matches(**char** guess) {

**return** guess == firstChar;

}

**public** **static** CodePeg getByChar(**char** c) {

**switch** (c) {

**case** 'R':

**return** ***RED***;

**case** 'G':

**return** ***GREEN***;

**case** 'B':

**return** ***BLUE***;

**case** 'Y':

**return** ***YELLOW***;

**case** 'C':

**return** ***CYAN***;

**case** 'M':

**return** ***MAGENTA***;

**default**:

**throw** **new** IllegalArgumentException("Invalid input: " + c);

}

}

}

Then create another enum for KeyPeg:

**public** **enum** KeyPeg {

***BLACK***, ***WHITE***;

}

Both of these classes should compile and you should have no errors in your project.

## Creating the main game

Create a new **class** (not an enum this time!), called Mastermind. We’ll create this in sections, first the main structure:

**import** java.util.\*;

**public** **class** Mastermind {

**static** Scanner *inputReader*;

**public** **static** **void** main(String[] args) {

Mastermind m = **new** Mastermind();

m.playGame();

}

}

The m.playGame() line will not compile as the playGame method does not exist. So, add a new method before the last code curly-bracket “}”.

**public** **void** playGame() {

**int** howManyPegs = 4;

**int** guessesRemaining = 12;

CodePeg[] allPegs = CodePeg.*values*();

Random random = **new** Random();

printInstructions(allPegs.length, howManyPegs, guessesRemaining);

*inputReader* = **new** Scanner(System.***in***);

// Set the secret code to 4 random colours (colours may repeat, e.g. 2

// blue pegs)

CodePeg[] secret = **new** CodePeg[howManyPegs];

**for** (**int** i = 0; i < howManyPegs; i++) {

secret[i] = allPegs[random.nextInt(allPegs.length)];

}

}

You can see that this won’t compile without printInstructions, so let’s add this, underneath playGame. Note that most lines wrap around in this document, but you should be able to work out where to start a new line though.

**private** **void** printInstructions(**int** howManyColours, **int** howManyPegs, **int** totalGuesses) {

System.***out***.println("Welcome to Mastermind.");

System.***out***.println("You have " + totalGuesses + " to guess the secret code of " + howManyPegs + " pegs");

System.***out***.println("Enter a guess by typing the first letter of the colour for your guess.");

System.***out***.println("The colours available are: Red(R), Green(G), Blue(B), Yellow(Y), Cyan(C) and Magenta(M)");

}

You should now be able to run your program (right-click Mastermind.java -> Run As -> Java Application, after which you can press Ctrl-F11 to re-run), it will print instructions, calculate the secret code and then terminate.

We now need a method to capture a guess from the codebreaker:

**private** CodePeg[] getNextGuess(**int** howManyPegs) {

**while** (**true**) {

System.***out***.print("Enter guess (type first letters of each colour): ");

// Get a line of input from the user and convert to upper case (so

// "r" becomes "R", etc.)

String input = *inputReader*.nextLine().toUpperCase();

// Now convert the letters to a ColouredPeg array, one peg for each

// letter

**try** {

CodePeg[] guess = **new** CodePeg[input.length()];

**for** (**int** i = 0; i < input.length(); i++) {

guess[i] = CodePeg.*getByChar*(input.charAt(i));

}

// Break out of the loop as we've got a valid guess

**return** guess;

} **catch** (IllegalArgumentException iae) {

// Print an error and ask for input again

System.***out***.println(iae.getMessage() + ". Try again!");

}

}

}

This captures a guess from the user. We want to test this out, so add the following logic to the bottom of the playGame method, just under where the secret is calculated.

// Main loop

**while** (guessesRemaining > 0) {

CodePeg[] guess = getNextGuess(howManyPegs);

// Work out how good the guess was and print the result

**if** (guess == **null**) {

System.***out***.println("Invalid guess.");

} **else** **if** (guess.length > howManyPegs || guess.length < 1) {

System.***out***.println("You can only guess between 1 and " + howManyPegs + " pegs");

} **else** {

// Now we know our guess was valid

// TEMPORARY CODE START

System.***out***.print("Guess was: ");

**for** (CodePeg peg : guess) {

System.***out***.print(peg + " ");

}

System.***out***.println();

guessesRemaining--;

System.***out***.println(guessesRemaining + " guesses remaining");

// TEMPORARY CODE END

}

}

This code should compile and you can now make lots of guesses. None of them will work, of course, because we haven’t written the code to calculate the value of the key pegs and show them to the codebreaker.

Add a new method to calculate the key peg result:

**private** List<KeyPeg> calculateResult(CodePeg[] secret, CodePeg[] guess) {

List<KeyPeg> result = **new** ArrayList<KeyPeg>();

**boolean**[] checked = **new** **boolean**[secret.length];

// Check to award black pegs

**for** (**int** i = 0; i < guess.length; i++) {

**if** (guess[i] == secret[i]) {

result.add(KeyPeg.***BLACK***);

checked[i] = **true**;

}

}

// Check to award white pegs

**for** (**int** i = 0; i < guess.length; i++) {

**for** (**int** j = 0; j < secret.length; j++) {

**if** (!checked[j] && (guess[i] == secret[j])) {

checked[j] = **true**;

result.add(KeyPeg.***WHITE***);

}

}

}

**return** result;

}

Read through this method and ask questions if you don’t understand what it’s doing. Finally, remove the temporary code from the playGame method (everything between and including the comments TEMPORARY CODE) we wrote in playGame and replace it with a call to playGame to use the new calculateResult method. This code goes inside the “else” block at the bottom of playGame.

// Calculate how many black and white pegs should be awarded

List<KeyPeg> result = calculateResult(secret, guess);

// Count how many black pegs there are, if 4 then codebreaker wins!

**int** blackPegCount = Collections.*frequency*(result, KeyPeg.***BLACK***);

**if** (blackPegCount == 4) {

System.***out***.println(

"Congratulations! You cracked the code with " + guessesRemaining + " guesses remaining.");

// Jump out the game loop

**break**;

}

// Otherwise, print out the key pegs

System.***out***.print("Result: ");

**for** (KeyPeg peg : result) {

System.***out***.print(peg + " ");

}

System.***out***.println();

// One less guess remaining!

guessesRemaining--;

// If you've run out of guesses, end the game and print the code.

**if** (guessesRemaining == 0) {

System.***out***.println("Bad luck, you didn't guess the code. It was: ");

**for** (CodePeg peg : secret) {

System.***out***.print(peg + " ");

}

} **else** {

System.***out***.println(guessesRemaining + " guesses remaining");

}

That’s it! You should now be able to play against the computer and see if you can break the code!