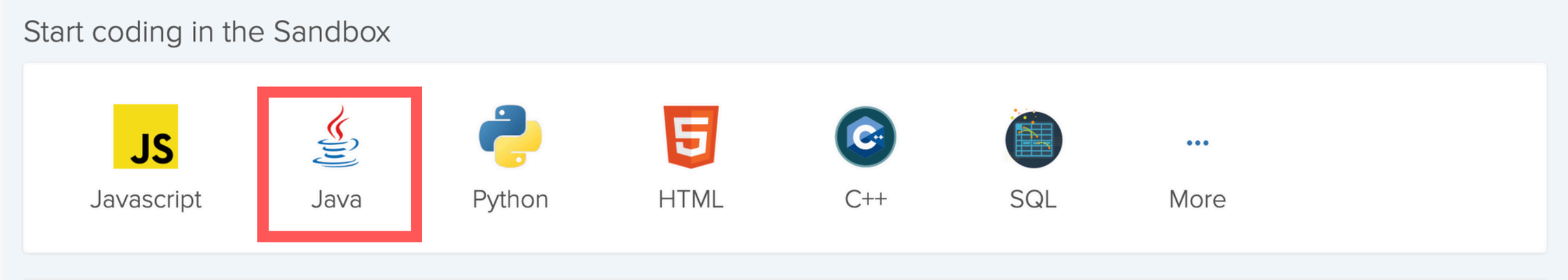
HANGMAN

WL HACK CLUB

CODEHS

- Go to **CodeHs**
- Create a new **HTML Sandbox Project**



SETUP

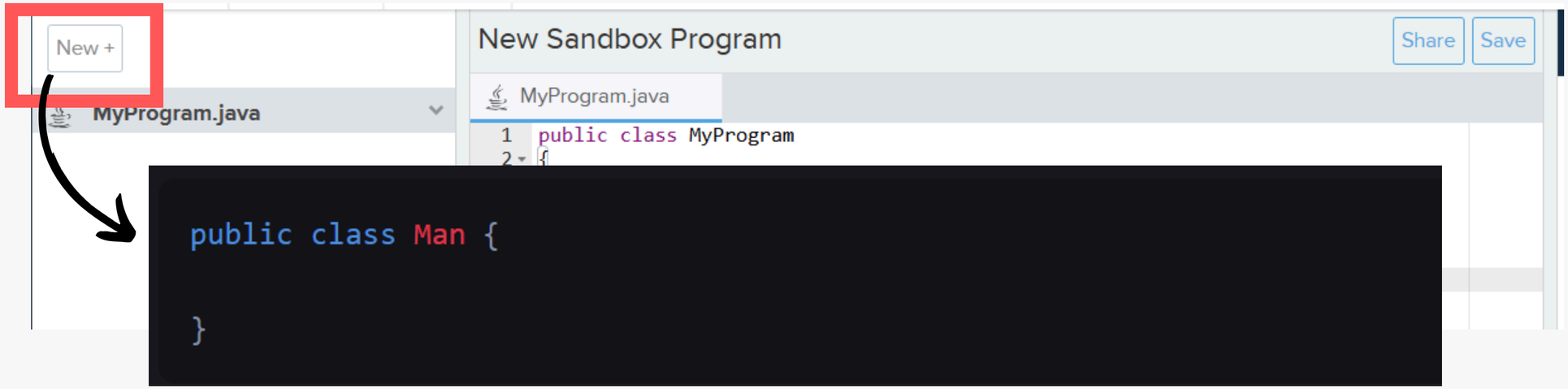
- We will be creating two classes
- Man class will be handling everything related to the hangman
    - drawing of the hangman, and keeping track of whether it’s dead or alive

- Separating the code into two classes to be easier

Fully Constructed “Hangman” will look like this:

MAN.JAVA

- Start by creating a file called ***Man.java*** file. Then, declare a class called **Man**



- ***Public*** *keyword so that our Man can be accessed by other classes too*

MAN.JAVA

- Let’s create some fields (variables specific to the class)

```
public class Man {
  static final int MAX_INCORRECT = 6;
  int numIncorrect;
  char[] body;
}
```

- **MAX_INCORRECT** a *constant*, so we use the static and final modifiers to indicate that it will remain the same for all instances of Man and never change

- **numIncorrect** is an integer that keeps track of the number of incorrect guesses.
- **body** is an Array of characters that stores the body parts of the hangman.

CONSTRUCTOR

- **Constructor** - a special method that is used to initialize objects
- Man is 3 characters wide and 3 characters tall
- create an array of characters of that size
- **\n** which represents a line break

```
public class Man{
  // Instance variables go here...
public Man() {
  // Initialize the Man object
  body = new char[] {' ', ' ', ' ', '\n', ' ', ' ', ' ', '\n', ' ', ' ', ' ', '\n'};
  numIncorrect = 0;
}
}
```

ISALIVE()

- At the bottom of your **Man** class, add:

```
public boolean isAlive() {
  return numIncorrect < MAX_INCORRECT;
}
```

- **boolean** - a data type that stores whether something is true or false
- return true or false, depending on whether or not the current number of incorrect guesses is less than the maximum allowed incorrect guesses.

TOSTRING()

- At the bottom of your **Man** class, add:

```
public String toString() {
  return new String(body);
}
```

- Java allows us to combine an array of characters and print them out as a string using the built-in String class

HANG()

- we need to hang different parts of the man according to how many incorrect guesses the user has so far

- **switch statement** - Pass in a variable to the switch statement, and then write specific code for each of the different cases or possibilities of that variable, using the keyword case

- \ alone is a special Java character. Have to add an extra \ so that it prints out an actual backslash

```
public void hang() {
  numIncorrect++;
  switch(numIncorrect){
        case 1:
          body[1] = 'O';
          break;
        case 2:
          body[5] = '|';
          break;
        case 3:
          body[4] = '\\';
          break;
        case 4:
          body[6] = '/';
          break;
        case 5:
          body[8]='/';
          break;
        case 6:
          body[10] = '\\';
          break;
  }
}
```

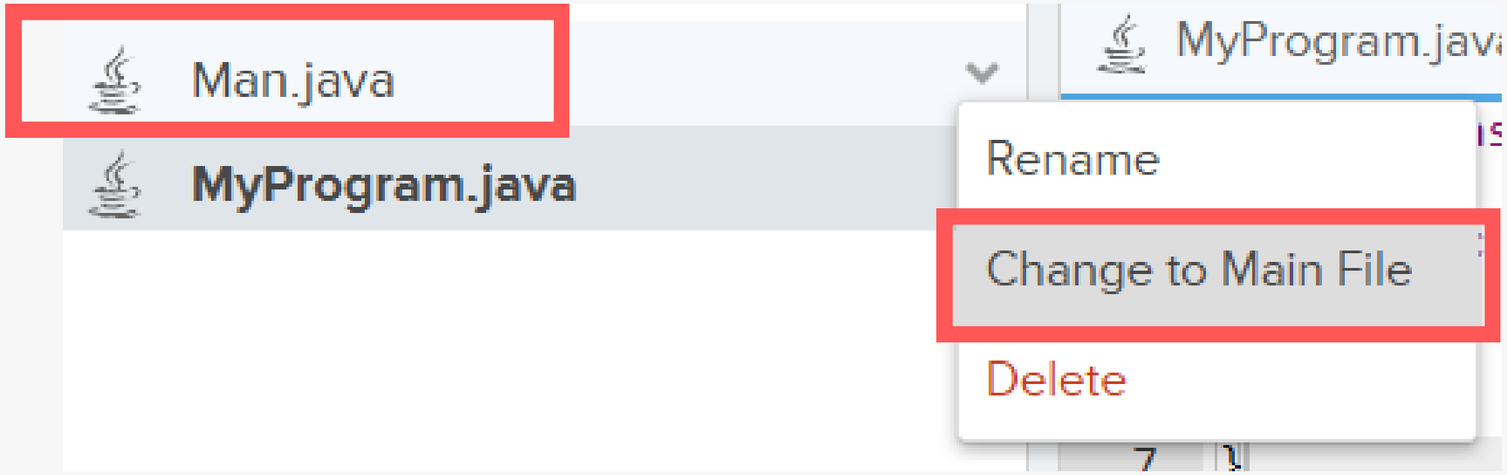
MAN TEST

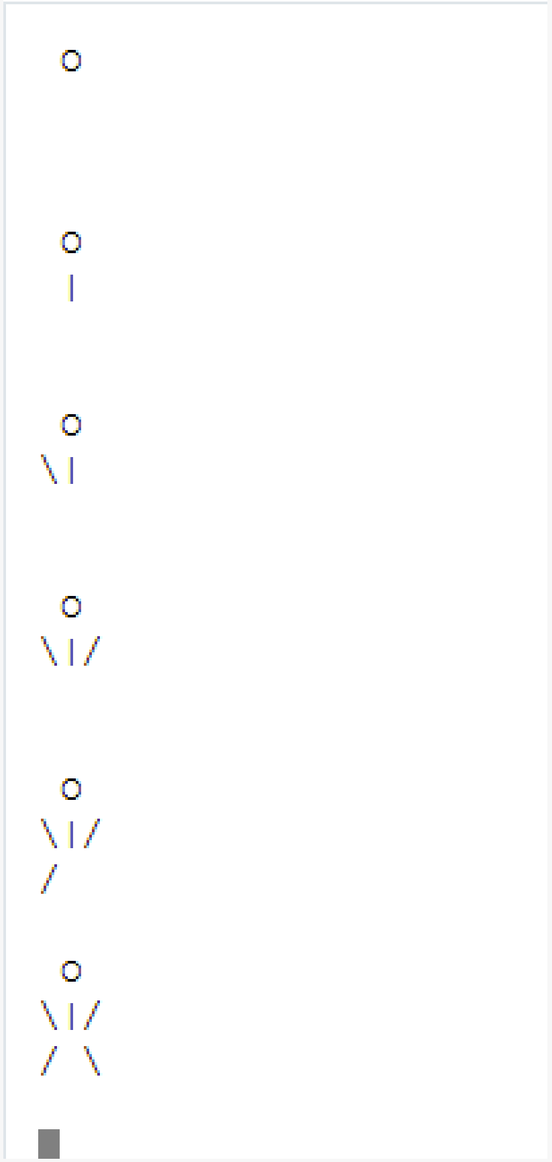
- we’re done with the Man class!
- At the bottom of your **Man** class, add:

```
public static void main(String[] args) {
  Man m = new Man();
  for(int i=0; i<Man.MAX_INCORRECT; i++) {
    m.hang();
    System.out.println(m);
  }
}
```

HANG()

- Change **Man.Java** to the main file & run
- You should see all the different stages of the man being hanged before dying!





MAIN.JAVA

- For the game logic, all of our code will live inside the main method in the Main class
- First off, we want to print a welcome message, just so people know they are playing hangman

```
System.out.println("Welcome to the ASCII Version of Hangman!");
```

- Change this file back to the main file

CONSOLE OBJECT

- Create a **Console** object and make that Console object read in a password which hides the word when typing
- Store that password into an Array of characters and convert each letter into uppercase
- Use the letters array's length property (using **.length**) so that we can go through each element of the array.

```
Console c = System.console();
char[] letters = c.readPassword("Please enter a secret word: ");
for(int i=0; i<letters.length; i++) {
  letters[i] = Character.toUpperCase(letters[i]);
}
```

IMPORTS

- let’s quickly add two import statements at the top of our code
- need to import **Console** and **Scanner** so that our program can use their functionalities.

```
import java.io.Console;
import java.util.Scanner;

public class Main {
        ...
}
```

DASHED LINE

- need to create another Array of characters of same length as letters and have all the letters be underscores with the help of a for loop
- add the following inside the **Main** class (beneath our previous for loop).

```
char[] puzzle = new char[letters.length];
  for(int i = 0; i < puzzle.length; i++) {
    puzzle[i] = '_';
}
```

GAME LOOP

- Need to create a Man object to hang and a Scanner object to accept user input for letters
- Add the following code right below our previous snippet.

```
Man m = new Man();
Scanner s = new Scanner(System.in);
```

GAME LOOP

How is Hangman played?

- Someone guesses a letter.
- If it's right, then you replace all the letters of the word that corresponds to the correctly guessed letter. If it's wrong, then you hang the man once.
- Once all the letters are guessed, they win; otherwise, you win. We can carry over this logic to Java and use a while loop.
- Each iteration of the while loop is a turn for the game, carrying out different actions based on the guessed letter.
- The while loop should only be run when the man is alive, so we set our condition to **m.isAlive().**

```
while (m.isAlive()) {
        //TODO: Add main game logic here
}
```

PRINT GUESS SO FAR

- Create **turns**
- Prompt the user to enter in a character
- Show remaining character with underscore

```
System.out.println("Puzzle to solve: ");
for (int i = 0; i < puzzle.length; i++) {
  System.out.print(puzzle[i] + " ");
}
System.out.println(); //Line of space
```

GUESS

How can we guess? **What if someone types in a full word instead of a letter?**

- Use the **scanner** from earlier to receive user input
- Only accept first character of the letter
- .charAt(0) takes the **first letter only**

```
System.out.print("Please guess a letter: ");
char guess = s.nextLine().toUpperCase().charAt(0);
```

GUESS

- Check if the word contains user guess
- Declare a **Boolean** to check true or false
- **containsGuess**
  - Set to false
  - Go through **letters(answer)** & compare
  - If we find a match, then true! Replace underscore with letter
- Unmatched characters are still underscores

```
boolean containsGuess = false;
for (int i = 0; i < letters.length && !containsGuess; i++) {
  if (letters[i] == guess) {
     containsGuess = true;
     for (int j = 0; j < letters.length; j++) {
        if (letters[j] == guess) puzzle[j] = guess;
     }
     break;
  }
```

WRONG GUESS

- If no matching character is found, start **hanging the man**
- Can’t just use **hang()** because this method is not in the Main class
- Instead, type **m.hang()**.
- **m** is from the new man we declared earlier.
- Make sure the print the hang man to see the progress

```
if (!containsGuess) m.hang();
```

```
System.out.println(m);
```

CHECK WINNER

How can we tell when the player has won?

- The word must be complete
  - (loop through puzzle)
- If there are no underscores -> Game complete!
- Boolean **checkUnderscore** turns to **true** if it encounters an underscore

```
boolean checkUnderscore = false;
for (int i = 0; i < puzzle.length; i++) {
  if(puzzle[i] == '_') checkUnderscore=true;
}
if(!checkUnderscore) break;
```

PRINT WINNER

- Write a **print statement** outside the while loop bracket
- If the man is **alive**, that means Player 2 has guessed correctly.
- If the man is **dead**, that means Player 2 **failed** to guess correctly, therefore Player 1 wins
- Use simple **if-else statmemnt** to check conditions and print different sentences

```
if (m.isAlive()) System.out.println("Success!  Player 2 wins!");
else System.out.println("Game over!  Player 1 wins!");
```

FINISHED!