

Guessing Game with Objects

Objectives	Practice writing an object-oriented program using different classes for different parts of the game.
Starter Code	There is starter code on Github Classroom. Use link http://bit.ly/OOP2020-lab1
What to Submit	Submit your code to Github Classroom, as instructed in class

1. Guessing Game

A guessing game is played on the console like this:

```

Guess a secret number.
I'm thinking of a number between 1 and 100.
your guess? 32
Sorry, too small.
your guess? 64
Sorry, too large.
your guess? 48
Right! The secret number is 48.
    
```

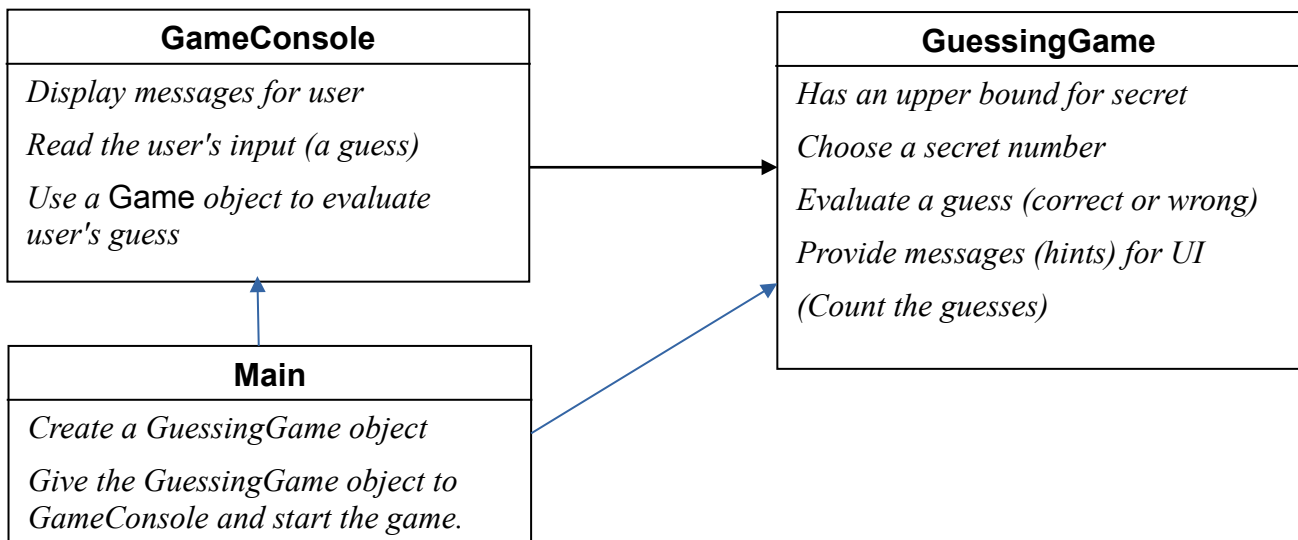
In object-oriented programs, we divide the problem into parts (classes) such that each class performs only one group of related **responsibilities**. We want classes and objects so that:

- (a) each class has only one or a few closely related responsibilities
- (b) classes are **simple** and easy to modify
- (c) a class provides methods that other objects need to do their job

In the guessing game, there are 3 sets of **responsibilities**:

1. Interact with the user: *print messages and read input until game is over*
2. Manage the game: *pick the secret number, evaluate a guess, give hint, decide when game is over.*
3. Run the game. *Called the "Application class" or "main class".*

We will define one class for each sets of responsibilities:



Application Design in UML

This is a UML class diagram for the classes in your program.

GuessingGame
-upperBound: int #secret: int
<<constructor>> +GuessingGame () #setMessage(msg: String) +getMessage(): String +getUpperBound(): int +guess(number: int): boolean +toString(): String

getMessage() returns a message based on most recent guess. The initial message is something like "I'm thinking of a number between 1 and NNN". After a guess, the message will be something like "Your guess is too small".

getUpperBound() returns the upper-bound for the secret number.

guess(number) submits a guess. guess() returns true if the guess matches the secret number, false otherwise.

GameConsole
+play(game: GuessingGame): int

Write a GameConsole to play a guessing game, which is given as a parameter to the play() method.

The play() method prints messages (from the GuessingGame), prompts the user for a guess, and submits the guess to the game. The Java method signature is:

```
public int play( GuessingGame game ) {
    ...
}
```

Main
<u>main</u> (String[] args): void

The Main class contains no attributes and only a static void main(String[] args) method. main does this:

```
public static void main(String[] args) {
    1. create a GuessingGame object
    2. create a GameConsole object
    3. call gameConsole.play( game )
    4. print the answer returned by play()
}
```

1.1 Try the GuessingGame class in BlueJ Codepad

You can create objects and call their methods interactively. You can verify that the GuessingGame works.

```
> game = new GuessingGame();
> game.getMessage()
"I'm thinking of a number between 1 and 100."
> game.guess(50)
false
> game.getMessage()
"Your guess is too small."
```

1.2 Write the GameConsole Class and Test It

Write this class in the default package. This class has only 1 method: **play**.

The **play** method should:

- a) describe the game to the user (game.toString).
- b) Use a loop to:
 - print a message from the game
 - ask the user to guess the answer.
 - call game to evaluate the user's guess.
- c) Return the correct answer when game.guess(number) returns true.
- d) Only call game.guess () one time for each value input by the user. Later you will add a counter to the game to count how many guesses the user makes!.

4. Write **Javadoc** comments for the GameConsole class and the **play** method.

```
/**
 * The play method plays a game using input from a user.
 * @param ...
 * @return ...
 */
public void play(GuessingGame game)
```

1.3 Write a Main class to create objects and start the game

1. Create a class named Main with a static **main** method.

The job of the main method is to create objects, connect user interface to the game, and start the user interface. **main** is a **static** method. **main** should be *simple (no application logic)*.

```
/** create objects and start the game */
public static void main( String [] args ) {
    GuessingGame game = new GuessingGame( );
    GameConsole ui = new GameConsole( );
    int solution = ui.play( game );
    //TODO print the solution
}
```

2. Test everything.

Problem 2: Add another Constructor to GuessingGame

The GuessingGame is too easy. We'd like to be able to create game with any upper bond for the secret.

Lab1: Guessing Game with Objects

Java classes can have more than one constructor, provided that each constructor has different parameters. This is different from Python -- Python only allow ones constructor, but parameters can have default values.

1. Add a constructor with an `int upperBound` parameter to `GuessingGame`.

```
/** A Constructor for new Game objects.
 * @param upperbound is the upper limit for the secret number
 */
public Game(int upperbound) {
    this.upperBound = upperbound; // save the upper bound
    //TODO complete rest of the constructor
}
```

Now we can specify an upper bound when we create a game:

```
game = new Game(1000000); // game with a secret between 1 and 1000000
```

2. Now your `Game` has **two constructors**: a *default* constructor and *parameterized* constructor.
3. The constructors contain **duplicate code** (two codes that do the same thing). **Duplicate code is bad.** You can eliminate the duplicate code by having one constructor invoke the other constructor. **Ask the TA to explain how.**
4. Modify **Main** to use the new constructor. Now you can control the game difficulty.

Set the Game Difficulty from the Command Line

Java lets you pass *arguments* from the command line to a Java program.

The command line arguments are Strings (even if the values are numbers) and are stored in the `String` array. These command line arguments are put in the `args[]` array that is a parameter to `main`. Here is how to get a number from the `args` array:

```
public static void main( String [] args )
{
    int bound = 20; // default value
    if ( args.length > 0 ) bound = Integer.parseInt( args[0] );
}
```

Problem 3: Add a Counter to Count guesses

1. Add a counter to your guessing game class to count how many guesses the user makes.

Add a counter to the `GuessingGame` class, not the `GameConsole` class.

2. Provide an *accessor* method for the counter named `int getCount()`.
3. Write good Javadoc for the method.
4. Modify the `Main` class so that it prints how many guesses the user made (call `getCount`).

Problem 4: Write a GameSolver Class to Automatically find the Secret

Write a `GameSolver` class that plays any `GuessingGame` and returns the answer (the secret).

This class has just one method: `int play(GuessingGame game)`. Just like `GameConsole`.

1. The `GameSolver` should **not print anything** -- just return the solution.
2. Use an efficient algorithm. Don't use a loop to guess all possible numbers.

Lab1: Guessing Game with Objects

Exception: if your GameSolver determines that there is no solution or the game is impossible, you can print a message on the console. In this case, return -1.

```
/**
 * Automatically find the secret to any GuessingGame.
 */
public class GameSolver {
    /**
     * Play a GuessingGame and return the solution.
     * The game must provide messages (getMessage) containing the
     * phrase "too small" if a guess is too small or "too large" if
     * a guess is too large, for efficient solution.
     *
     * @param game is the GuessingGame to solve
     * @return //TODO what does it return?
     */
    public int play(GuessingGame game) ...
}
```