Guessing Game with Objects

Objectives	1. Practice writing an object-oriented program using different classes for different parts of the game. 2. Write code that is portable. Your application must be able to work with parts written by other students, e.g. your user interface can play someone else's game object.
Tasks	1. Write a guessing game class that extends NumberGame. 2. Write a user interface class name GameConsole and a Main class. 3. Implement the program and write documentation.. Test it often! 4. Commit your code to Git often. Don't wait 'til everything is done! 5. Copy a game class from another student. Verify that your application works with their guessing game class (class that extends NumberGame). 6. Write a class named GameSolver that can solve any NumberGame, provided that game.getMessage() returns strings containing "*too small*" or "*too large*" to help your solver find the solution. 7. Submit your code to Github, as instructed in class.

Design of the 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 related tasks or related **responsibilities**. Try to design classes and objects so that:

(a) each class has only one or a few closely related responsibilities

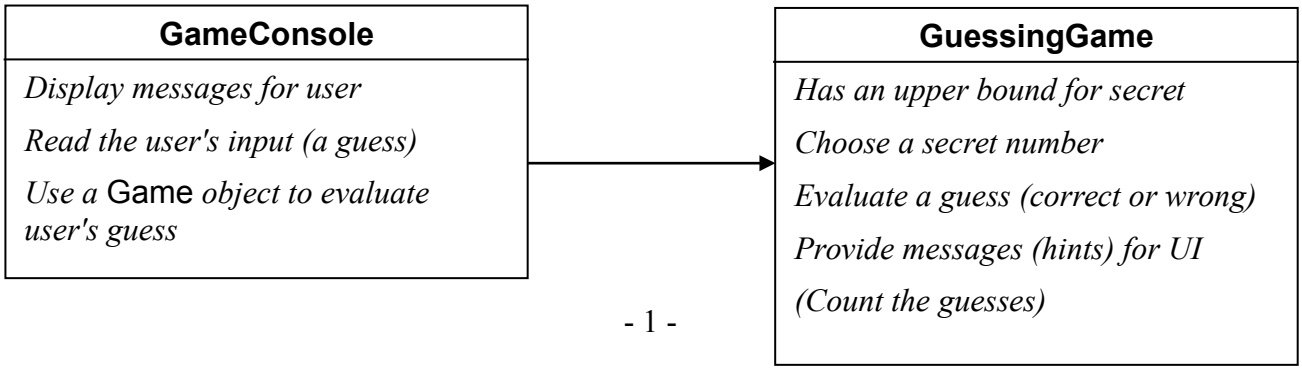
(b) classes are **simple** and easy to modify

(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. Create objects and start the game. *Called the "Application class" or "main class".*

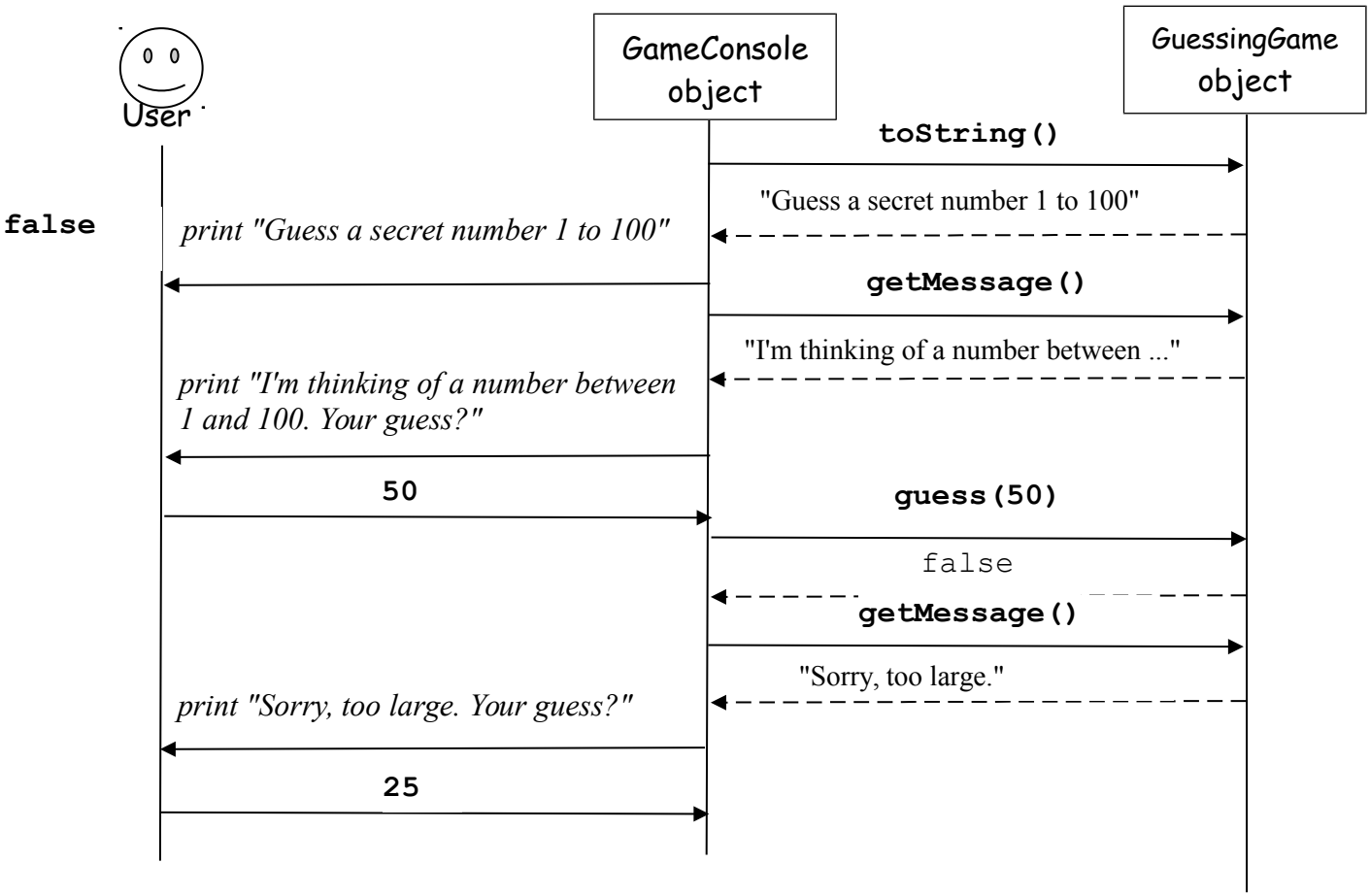
We will define one class for each sets of responsibilities:



Main
Create a GuessingGame object *Give the GuessingGame object to GameConsole and start the game.*

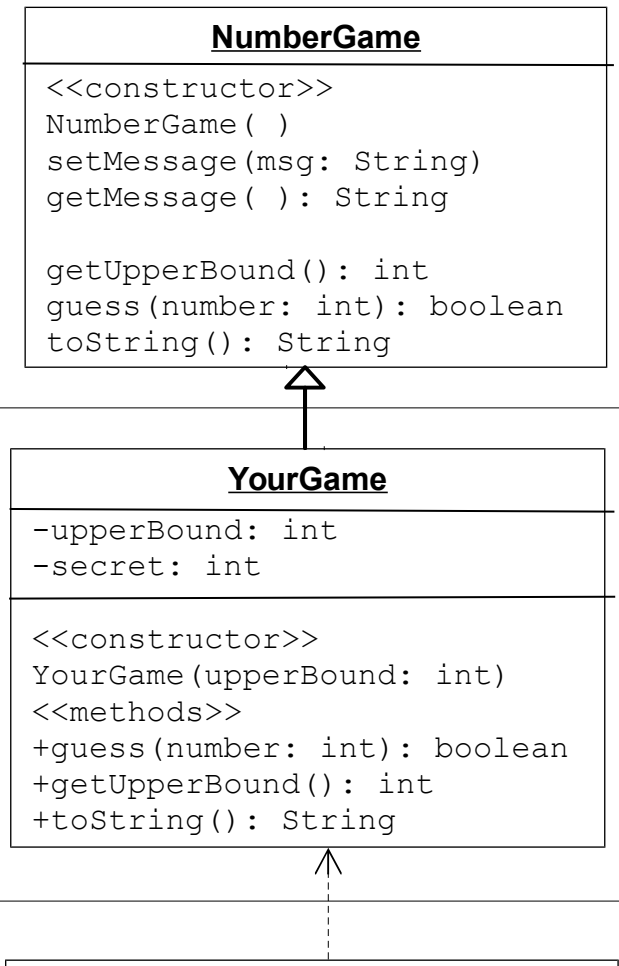
The arrow shows that the GameConsole needs to "know" about a Game object, because GameConsole has to "ask" the Game if a user's guess is correct, and "ask" for a hint.

The GameConsole and GuessingGame objects work together when you run the program. Here is an ***example***:



Application Design in UML

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



This is the *superclass (base class)* for all games. The reason for this class is to enable *polymorphism*, so your code can play other students' game.

Your game should override these methods, but don't override setMessage and getMessage.

YourGame `extends NumberGame` and implements a guessing game.
You must write:
a) constructor with an upperbound for the secret
b) guess(value) - evaluate user's guess
c) getUpperBound()
d) toString() to describe the game
Use the setMessage() method to set hints for the user to help him play the game.
The messages can be *anything* but must contain the words "`too large`" if latest guess is too large, "`too small`" if guess is too small.

The user interface for playing the game.
The play method does:
a) print a description of the game (use toString)
b) print messages from the game (getMessage)
c) prompt user to guess the answer
d) give user's guess to the game object and tell the user if it guessed correct
e) return the correct answer

The play( ) method should be able to play *any game* that is a subclass of NumberGame. So you can play other students' games, too.

The main method should
a) create a `NumberGame` object
b) create a `GameConsole` object
c) tell `GameConsole` to play the game
d) print the solution returned by `play(game)`

Problem 1: Write a Guessing Game Class and Test It

1. Create a new class named *Yourname***Game**, e.g. ***FatailaiGame.***

```
import java.util.Random;     // for generating random numbers
/**
 * Game of guessing a secret number.  -- write a description
 * @author Fatalai Jon
 */
public class FatalaiGame extends NumberGame {
    /* properties of a guessing game */
   //TODO Declare variables for attributes of the game
```

You can use (but don't just copy) code from SampleGame to get started.

Your game should be a subclass of `NumberGame` ("`extends NumberGame`").

2. Define the attributes of your Game. It should have attributes for the secret (solution) and upperBound.

3. Write a **constructor with a parameter for the upper bound**. The constructor initializes a new game.

```
/**
 * Initialize a new game.
 * @param upperbound is the max value for the secret number (>1).
 */
public YourGame( int upperbound ) {
    (a) set the upperBound for the secret number
    (b) create a secret number
    (c) initialize the message to "I'm thinking of a number between 1 and XX".
}
```

4. Write the methods of the Guessing Game. You may also write **`private`** utility methods to simplify your code. The required methods are:

`boolean guess(int number)`	Evaluate a user's guess. Return true if it is correct, false otherwise. Also set a message to help the user.
`int getUpperBound()`	Return the upperbound for the solution to this game. Should be a positive integer.
`String toString()`	Describe the game. This is a general description, such as "Guess a secret number between 1 and 250".

5. The guess(number) method that evaluates a guess. It should call setMessage( ) to set a hint or other message. The message *must contain* "`too small`" if the guess is too small and "`too large`" if the guess is too large. Other than that, you can be creative or insulting (e.g. if the user guesses the same thing twice or a ridiculous guess).

Programming Hint: How to get a random integer

The `java.util.Random` class has methods to generate random numbers. The Random constructor accepts a `long` value called a *seed*. The *seed* initializes the random number generator so the random number sequene is unique. If you don't supply a seed (or use 0), the sequence of random number is always the same -- useful for testing.

```
// generate an unpredictable seed
long seed = System.nanoTime( );
Random rand = new Random( seed );
// get a random number between 0 and 9. Add 1 so the value is 1 - 10.
int value = rand.nextInt(10) + 1;
```

Problem 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 and 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 the game is going to count how many guesses the user makes!.

a result message *return the solution (the guessed secret)*

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 ...
 */
```

Problem 4: 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 ) {
      NumberGame game = new YournameGame( upperbound );
      GameConsole ui = new GameConsole( );
      int solution = ui.play( game );
   //TODO print the solution
}
```

2. Test everything.

Problem 5: Play another student's game

Your application should be able to play another student's game, by changing *only* one line of code in the main method.

Problem 6: 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. To enable *polymorphism*, also add `getCount()` to the `NumberGame` class.
In the `NumberGame` class, `getCount()` always returns 0.

4. Write good Javadoc for the method.

5. Modify the Main class so that it prints how many guesses the user made (call getCount).

Problem 7: Write a GameSolver Class to Automatically find the Secret

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

This class has just one method: **`int play(NumberGame game)`**.

The GameSolver should not print anything -- just return the solution.

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

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


}
```

Optional: Use Dialog Boxes instead of console input

This problem shows the benefit of using separate objects for separate responsibilities.

Instead of playing on the console, we can use dialog boxes. We won't need to rewrite the application! Just *replace* the `GameConsole` class with a new `GameDialog` class. We can *reuse* all other code from the application.

In a real project, you will often write a simple console input class to test your program logic, then write a graphical interface for really playing the game.

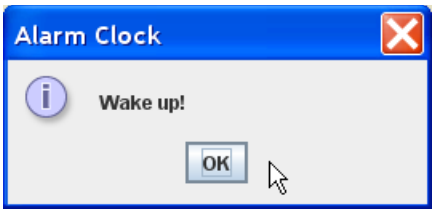
1. Create a new class named GameDialog with a **`play( )`** method, exactly like GameConsole.play().

2. Modify GameDialog to use **dialog boxes** instead of **`System.in`** and **`System.out`**.

How to Use Dialog Boxes

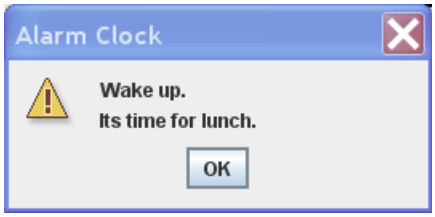
JOptionPane (in package javax.swing) contains many useful dialogs. Here are some examples. You can run these examples interactively in BlueJ.

Message Dialog




```
import javax.swing.JOptionPane;
String title = "Alarm Clock";
String message = "Wake Up.";
int type = JOptionPane.INFORMATION_MESSAGE;
JOptionPane.showMessageDialog( null, message, title, type);
```

Message Dialog with multi-line message:




```
message = "Wake up.\nIts time for lunch";
type = JOptionPane.WARNING_MESSAGE;

JOptionPane.showMessageDialog( null, message, title, type);
```

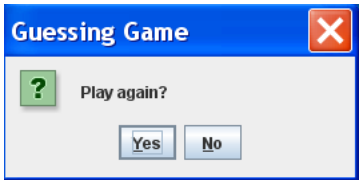
Input Dialog:

Use this to print a hint and ask user to guess a number.


```
title = "OOP Coffee Shop";
message = "We have all kinds of coffee.\n";
message += "What would you like to drink?";
type = JOptionPane.QUESTION_MESSAGE;
String reply = JOptionPane.showInputDialog(null,message,title,type);
if ( reply == null )
     /* user pressed Cancel */;
else orderCoffee( reply );
```

Test if user presses Cancel *The dialog returns a* `null`.

Confirm Dialog:




```
String title = "Guessing Game";
message = "Play again?";
type = JOptionPane.YES_NO_OPTION;
int opt = JOptionPane.showConfirmDialog(null, message, title, type);
if ( opt == JOptionPane.YES_OPTION ) /* user pressed "Yes" */;
```