GET SMART: WITH JAVA PROGRAMMING

YAMAN OMAR ALASHQAR

SYSTEM.OUT.PRINTLN(“WELCOME TO THIS COURSE”);

PREVENTING EXTENDING AND OVERRIDING

- Final is used to apply restrictions on class, method, and variable.
- The final class can't be inherited, final method can't be overridden,
- and final variable value can't be changed

PREVENTING EXTENDING AND OVERRIDING

- Standardization: Some classes perform standard functions and they are not meant to be modified e.g. classes performing various functions related to string manipulations or mathematical functions etc.
- Security reasons: Sometimes we write classes which perform various authentication and password related functions and we do not want them to be altered by anyone else.

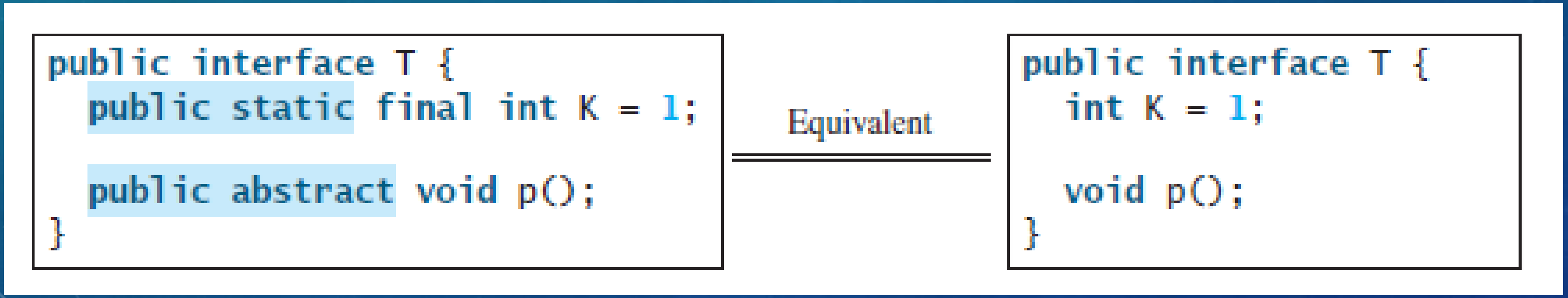
ABSTRACT CLASS

- An abstract class cannot be used to create objects.

- Some Rules:
  - An abstract class cannot be instantiated using the new operator.
  - An abstract class can contain abstract methods, which are implemented in concrete subclasses.
  - An abstract method cannot be contained in a nonabstract class

INTERFACE

- An interface is a class-like construct that contains only constants and abstract methods
- As with an abstract class, you cannot create an instance from an interface using the new operator
- All data fields are public static final and all methods are public abstract



```
public interface T {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

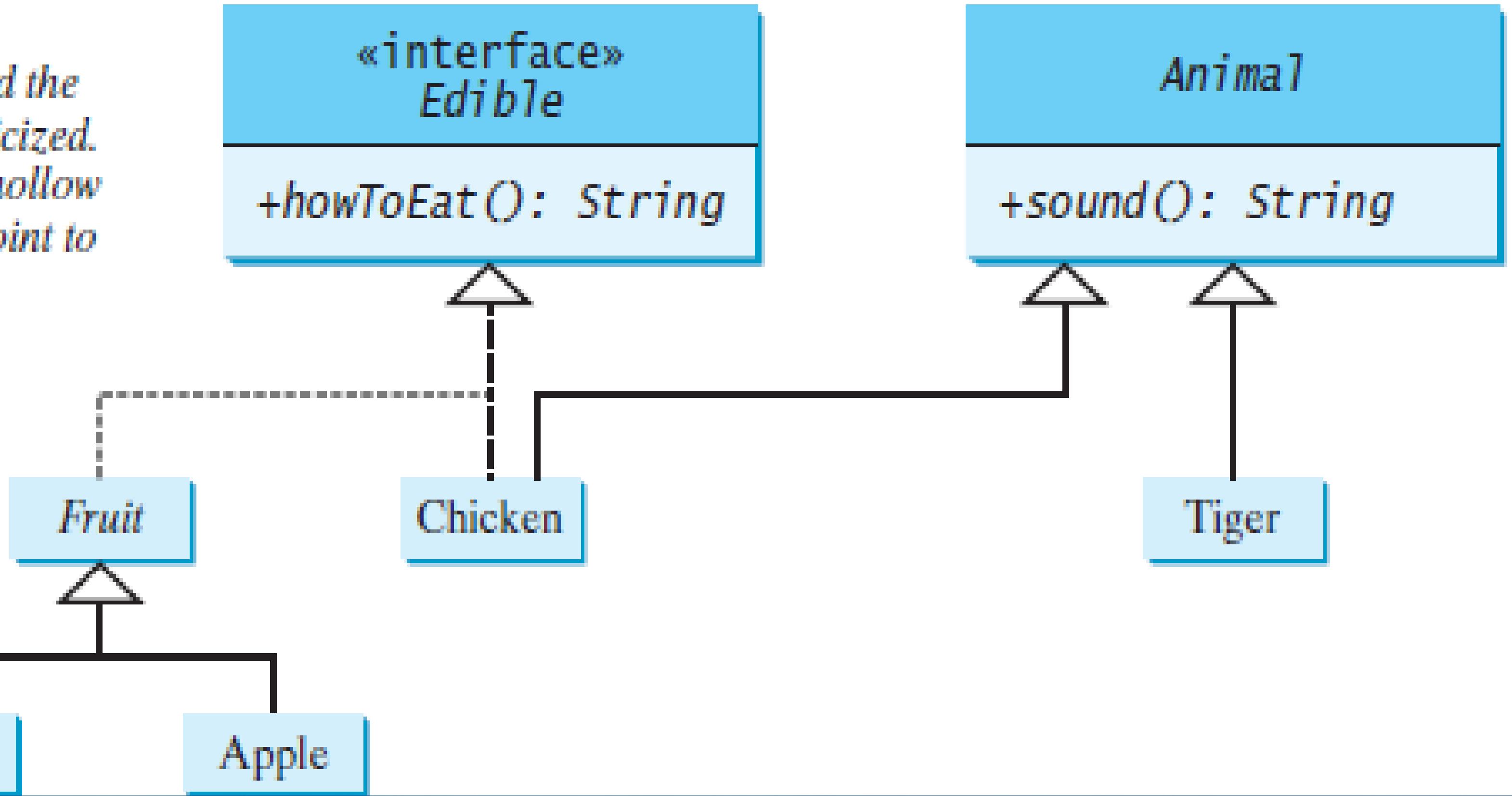
```
public interface T {
  int K = 1;

  void p();
}
```

INTERFACE

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .

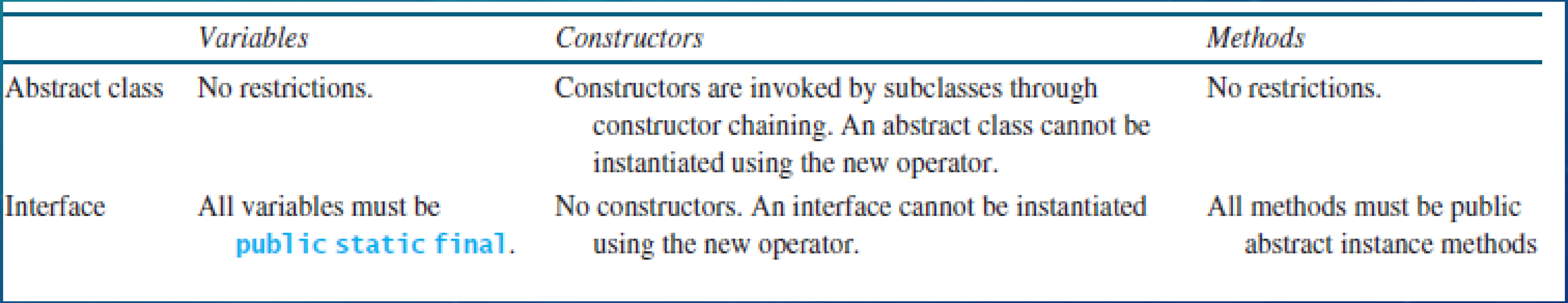
INTERFACE

Notation:
The interface name and the method names are italicized. The dashed lines and hollow triangles are used to point to the interface.



INTERFACES VS. ABSTRACT CLASSES

- A class can implement multiple interfaces, but it can only extend one superclass.
- An interface can be used more or less the same way as an abstract class



	Variables	*Constructors*	*Methods*
Abstract class	No restrictions.	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be `public static final`.	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

EXCEPTION-HANDLING

- Exception handling enables a program to deal with exceptional situations and continue its normal execution.

- In Java, runtime errors are thrown as exceptions. An exception is an object that represents an error or a condition that prevents execution from proceeding normally.

- If the exception is not handled, the program will terminate abnormally.

CATCHING EXCEPTIONS

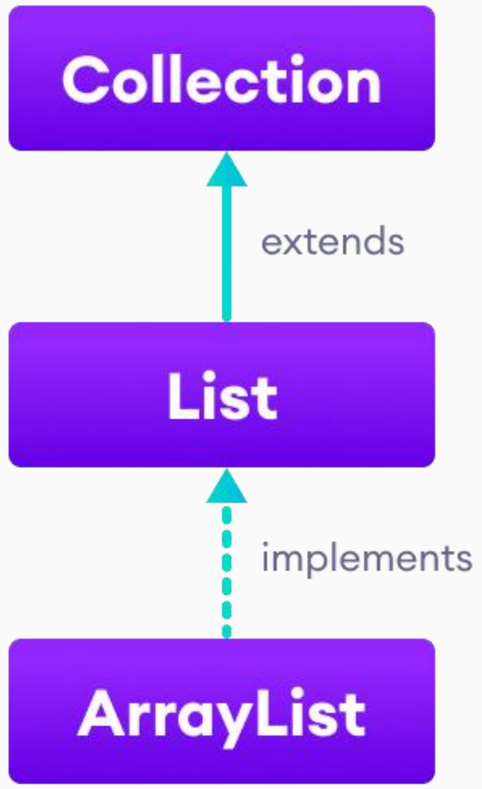
```
try {
	statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
	// handler for exception1;
}
```

JAVA ARRAYLIST

- The ArrayList class is a resizable array, which can be found in the java.util package.
- The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.
- ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object

Java ArrayList

- In Java, we need to declare the size of an array before we can use it. Once the size of an array is declared, it's hard to change it.
- To handle this issue, we can use the ArrayList class. It allows us to create resizable arrays.
- Unlike arrays, arraylists can automatically adjust its capacity when we add or remove elements from it. Hence, arraylists are also known as dynamic arrays.



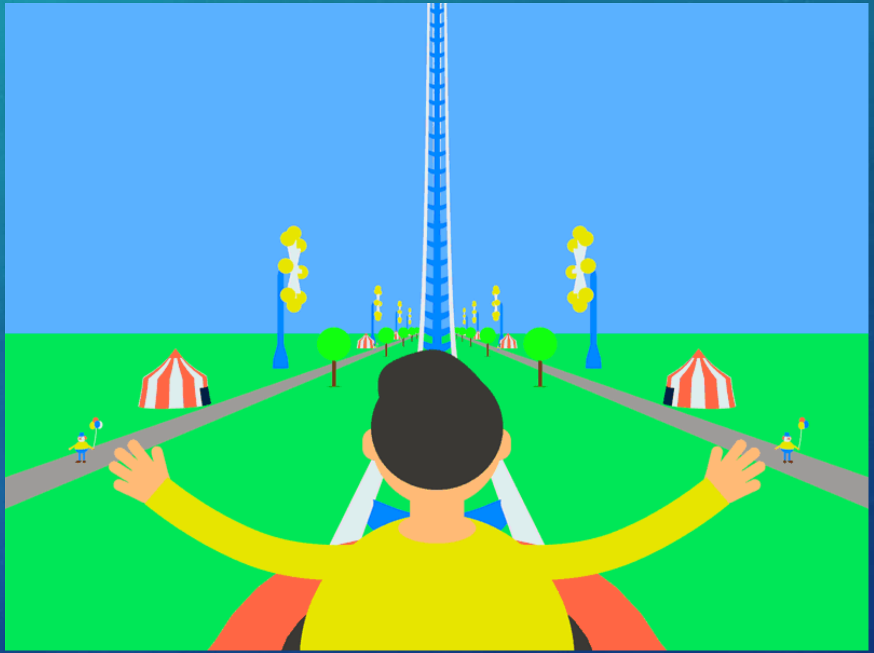
Basic Operations on ArrayList
ArrayList<String> languages = new ArrayList<>();

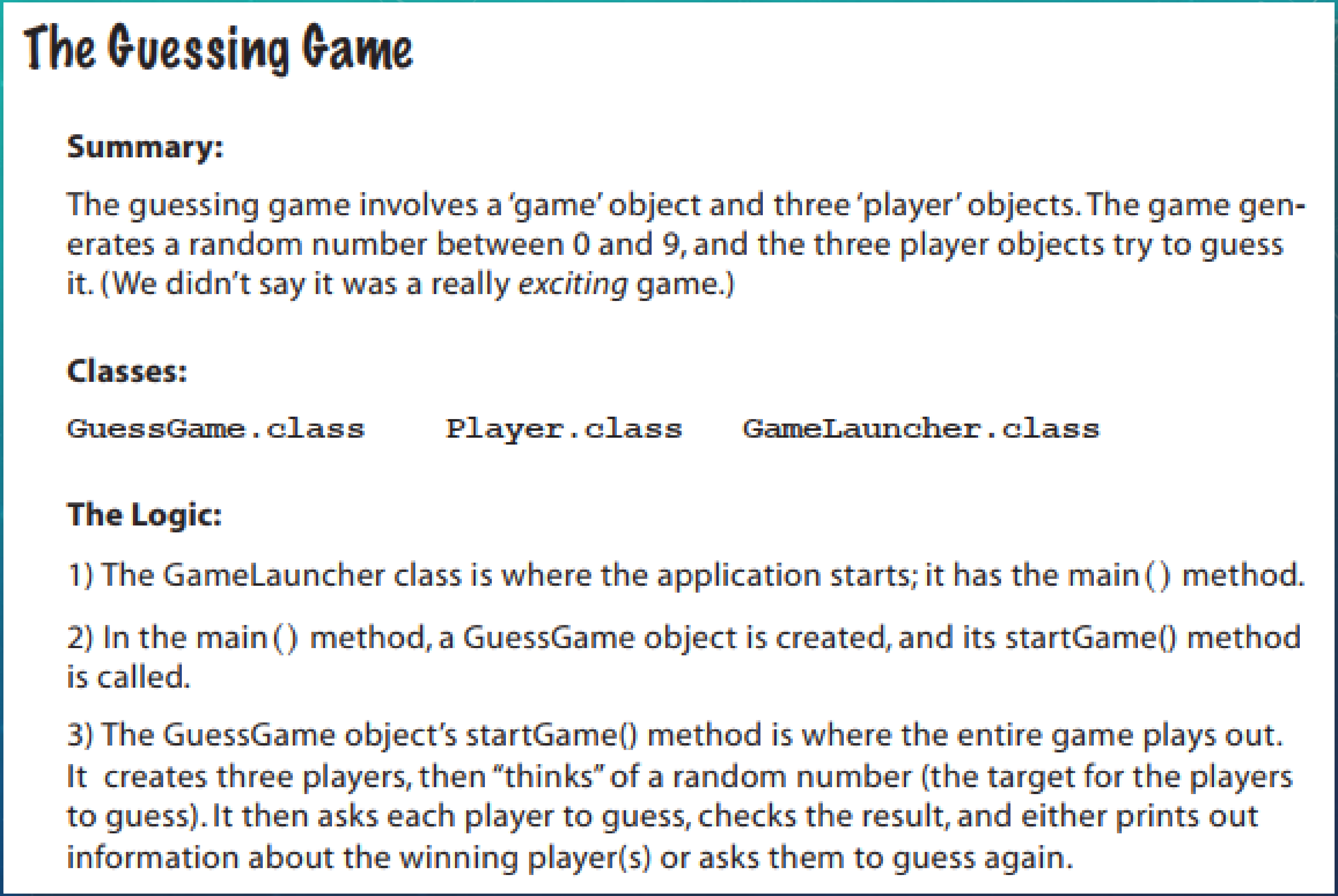
- Add Elements to an ArrayList
    - languages.add("Java"); // add() method without the index parameter
    - languages.add(1, "C++"); // insert element at position 1

- Access ArrayList Elements .get(1);
- languages.set(2, "JavaScript");
- .remove(2); OR languages.remove("Java");

CLASS ACTIVITYCOST

- Cost, TAX (20%), Start Time, End Time, numOfGamesPlayed, Name, age, height
- Entrance Fees: 0-3 Free (no games allowed) , 4-10 ($5), 11-50 ($7), 51+ (free)
- rollerCoaster() //AGE > 15
- wickedTwister() // AGE > 15
- ferriswheel()
- displayAllowedGames()
- isAllowed()





The Guessing Game

Summary:

The guessing game involves a 'game' object and three 'player' objects. The game generates a random number between 0 and 9, and the three player objects try to guess it. (We didn't say it was a really *exciting* game.)

Classes:

`GuessGame.class`   `Player.class`   `GameLauncher.class`

The Logic:

1) The GameLauncher class is where the application starts; it has the main() method.

2) In the main() method, a GuessGame object is created, and its startGame() method is called.

3) The GuessGame object's startGame() method is where the entire game plays out. It creates three players, then "thinks" of a random number (the target for the players to guess). It then asks each player to guess, checks the result, and either prints out information about the winning player(s) or asks them to guess again.