Review

Midterm Exam

CASTING

- In OO development, it's often useful to know exactly what type of object is contained in a variable.
    - In Java, ***type casting*** is implemented as <ClassName>, i.e.:

```
ArrayList whoKnows = new ArrayList();

ArrayList<Puppy> pups = new ArrayList<Puppy>();

Puppy[] finitePups = new Puppy[10];
```

    - This serves as a promise, and provides some security.

Composition of Classes

- If a class contains a reference to an object, this is a ***has-a relationship***.
    - Example: the `Company` class *has a* list of `Employee`s
        - Each employee is an `Employee` object, and `Company` *has a* reference to it.
- If a class is a type of another object (it inherits from another class), this is an ***is-a relationship.***
    - Example: the `Rectangle` class *is a* `Polygon`.
        - The `Polygon` is an abstract class, and `Rectangle` *extends* it.
    - Example: the `Employee` class *is a* `Payable`.
        - The `Payable` interface is *implemented* by `Employee.`

DECLARATIONS

- When creating a variable [which is not a primitive type], the **superclass** can be used in the declaration, and the **subclass** in the initialization.
  - *When might this be useful?*
  - For example, the following two lines of code are the same:

```
HourlyEmployee e1 = new HourlyEmployee(“Bob", "The Builder”, 10.0);

Employee e2 = new HourlyEmployee(“Bob", "The Builder”, 15.0);
```

The **SOLID** principles

- **SRP -** Single Responsibility Principle
- **OCP -** Open-Closed Principle
- **LSP -** Liskov Substitution Principle
- **ISP -** Interface Segregation Principle
- **DIP -** Dependency Inversion Principle

java.util.ArrayList

```
ArrayList<String> aList = new ArrayList<String>();
```

- Useful methods:
  - add an object to the ArrayList:
    ```
    add(Object o)
    add(int index, Object o)
    ```
  - get an object from the ArrayList:
    ```
    get(int index)
    ```
  - get the number of items in the ArrayList:
    ```
    size()
    ```
  - remove an object from an index in the ArrayList:
    ```
    remove(int index)
    ```

Polymorphism

Vehicle

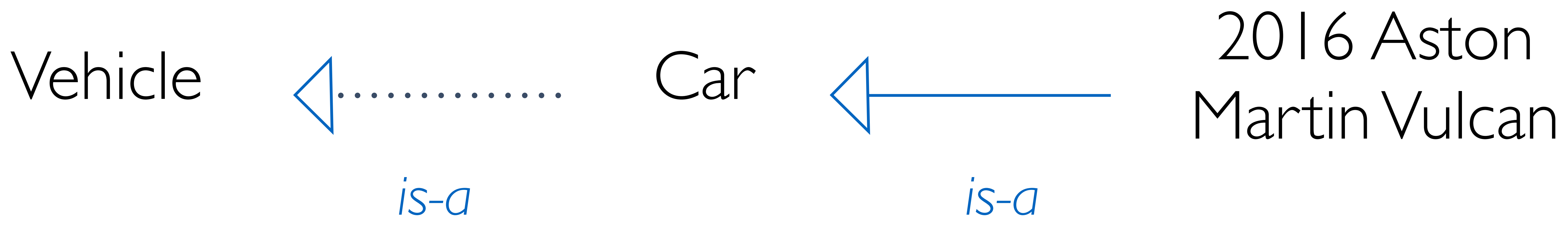
Car

2016 Aston Martin Vulcan

$\xrightarrow{\hspace{10em}}$

more specific

Polymorphism



Inheritance

$$\begin{tikzcd}
\text{Vehicle} & \text{Car} \arrow[l, dotted, "\textit{is-a}"'] & \text{2016 Aston Martin Vulcan} \arrow[l, "\textit{is-a}"']
\end{tikzcd}$$

Interface

Defines some functionality, which must be implemented in a subclass. **No implementation.**

Abstract class

Some implementation, but defines **some functionality** which *must be* implemented in a subclass.

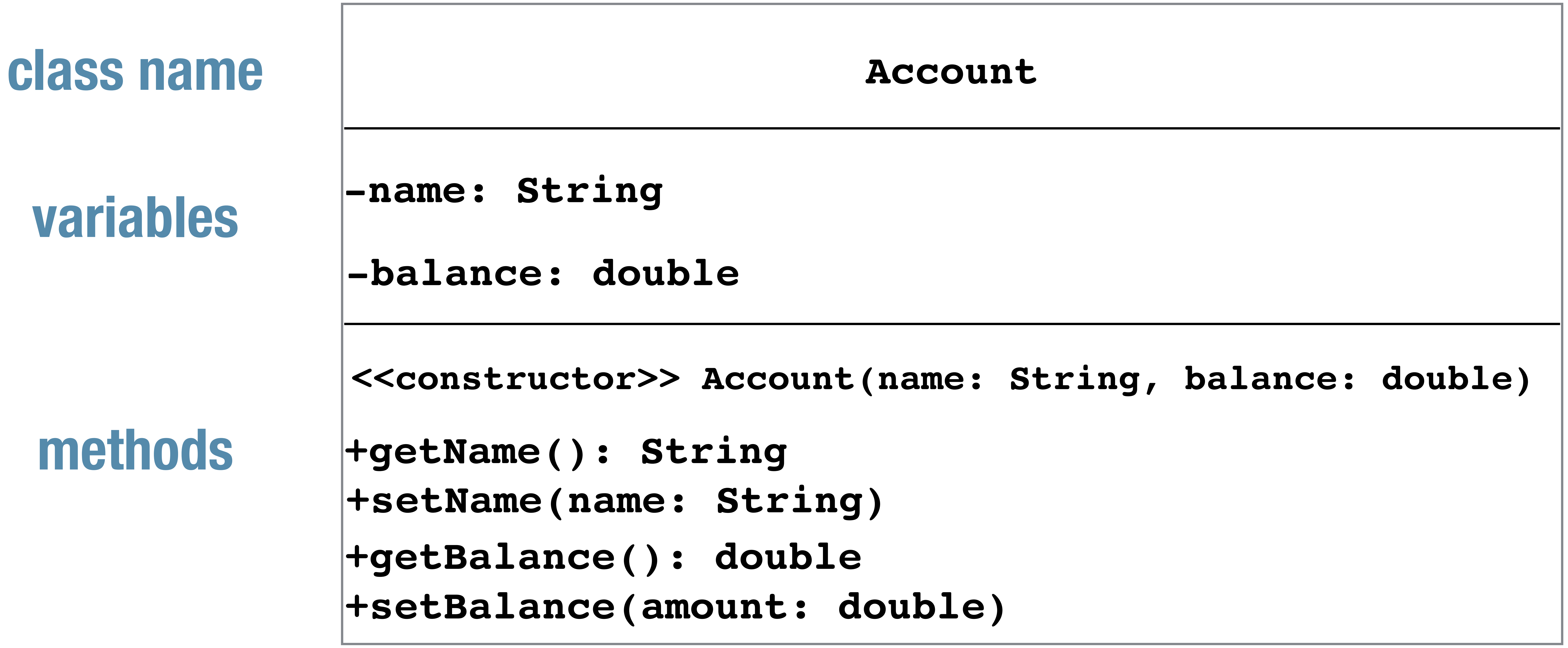
Class

Implementation code. *Implements* a superclass or *extends* an abstract class.

Inheritance

- Enables new objects to take on the properties of existing objects.
- **Superclass** = a class that is used as the basis for inheritance.
- **Subclass** = a class which inherits from a superclass.

UML



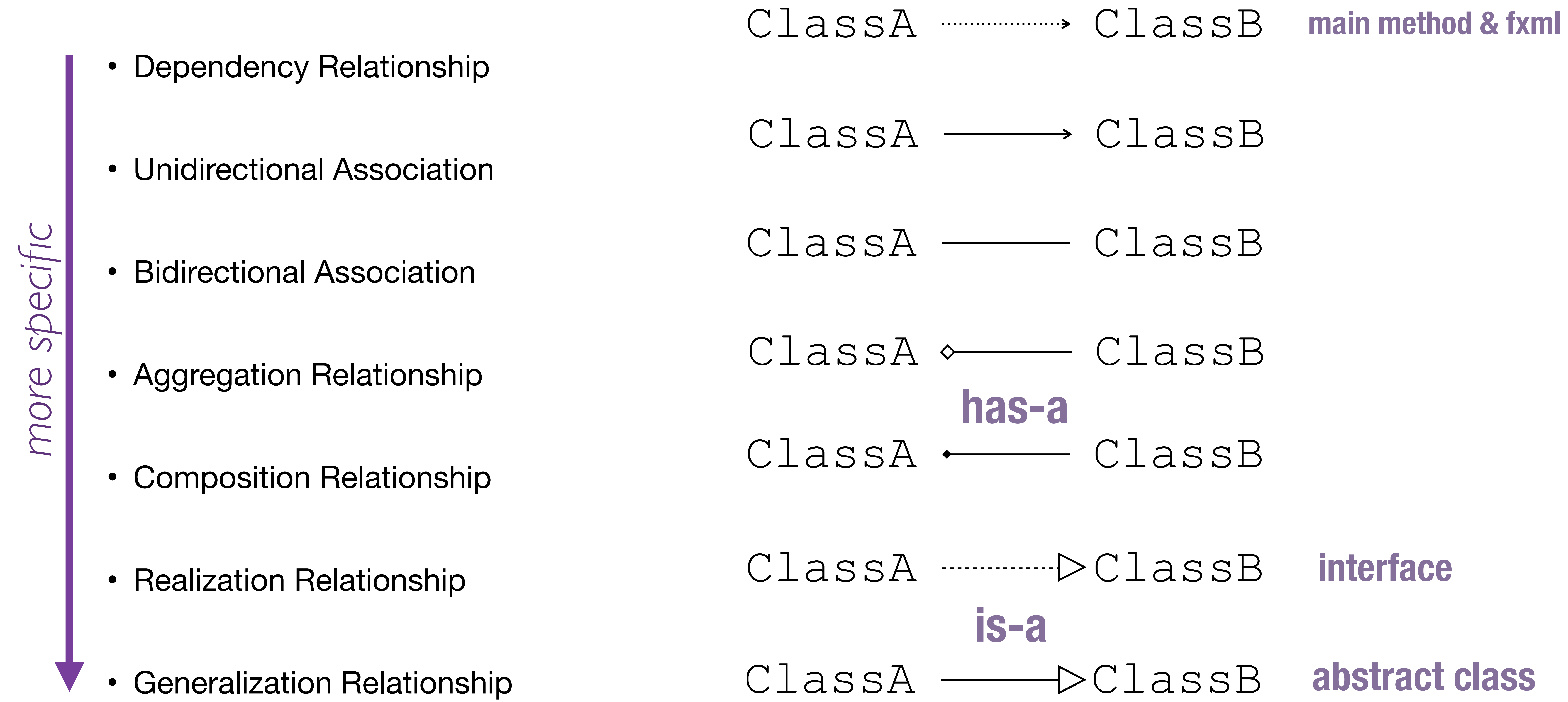
```
Account
```

```
-name: String
-balance: double
```

```
<<constructor>> Account(name: String, balance: double)
+getName(): String
+setName(name: String)
+getBalance(): double
+setBalance(amount: double)
```

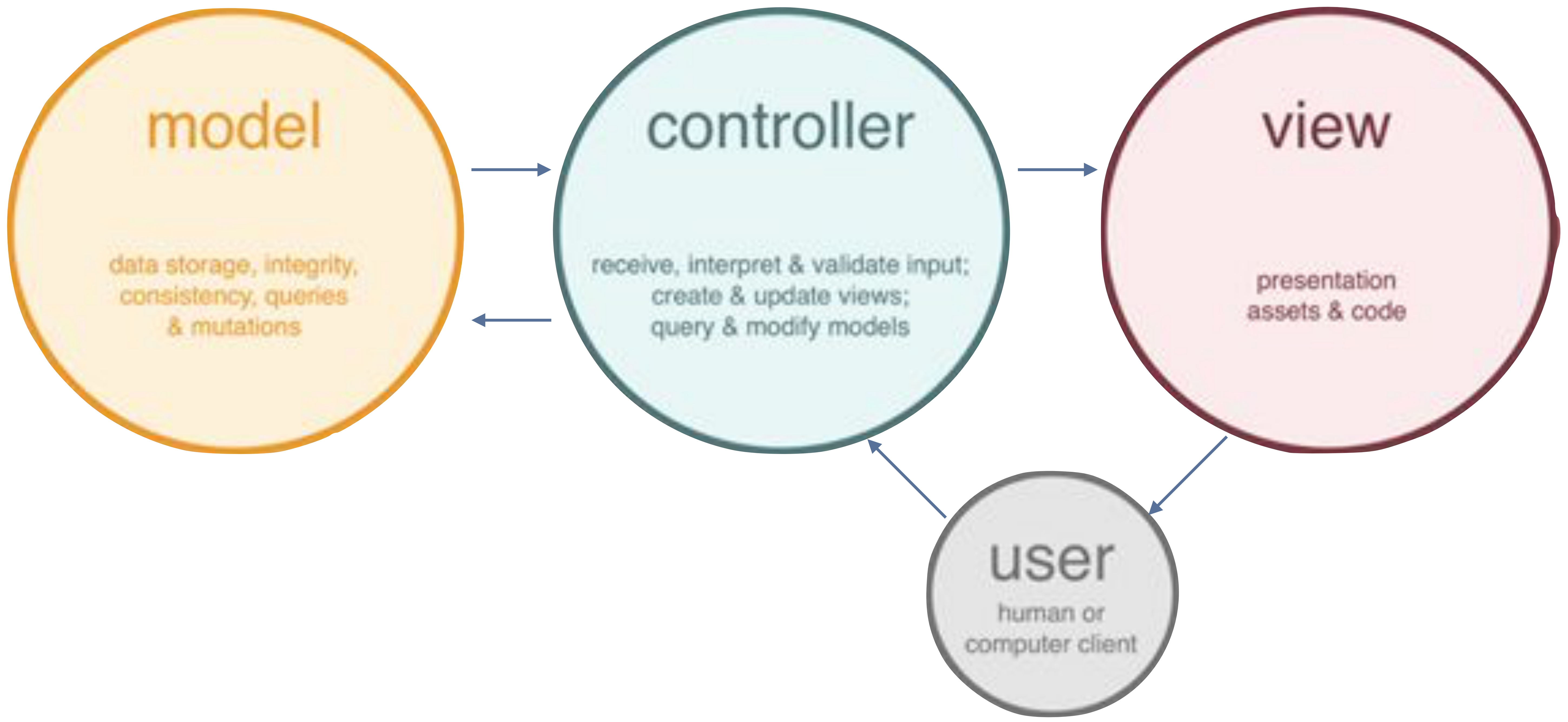
UML

Choose the most specific relationship!



- Dependency Relationship
- Unidirectional Association
- Bidirectional Association
- Aggregation Relationship
- Composition Relationship
- Realization Relationship
- Generalization Relationship

Model-View-Controller


model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

A basic JavaFX app in 6 steps

1. Create a new JavaFX project in Eclipse.
    - Creates a Stage with a main method - *to run your app*
    - Creates a Scene - *to hold the GUI components*
2. Create a new FXML document
3. Set up GUI components on your Scene
4. Connect your Scene to the Stage
5. Create a controller - *to listen and handle events*
6. Connect the controller to your application.

EBookReader review

Review Exercises

Midterm Exam

Methods

- Write a class method that takes one parameter, a String, and returns the number of vowels it contains.

Methods

- Write a class method that takes one parameter, a String, and returns the number of vowels it contains.

```
public static int vowels( String name ){

        // start a count of vowels (at 0)
        int count = 0;

        // count the vowels
        for( int i=0; i<name.length(); ++i)
             if(name.charAt(i)==‘a’ || name.charAt(i)==‘e’ || name.charAt(i)==‘i’ || name.charAt(i)==‘o’ || name.charAt(i)==‘u’)
                 count++;

      // return the count
      return count;
}
```

Complete Java Programs

- Write a complete Java program to prompt the user for a number, and print whether the number is odd or even.

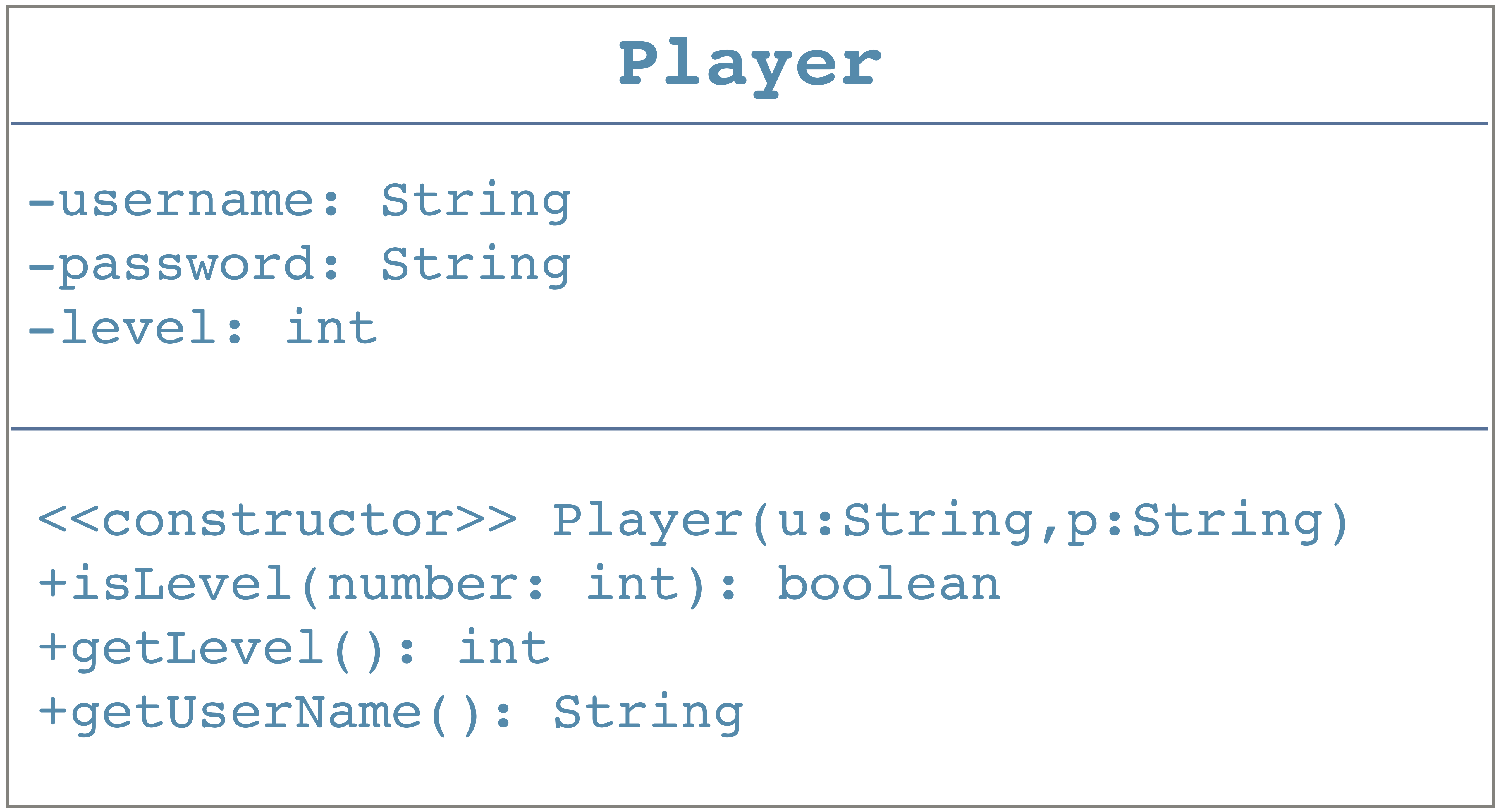
Complete Java Programs

- Write a complete Java program to prompt the user (from the console) for a number, and print whether the number is odd or even.

```
import java.util.Scanner;

public class ExampleQuestion{

   public static void main(String[] args){

         // prompt the user for a number
        Scanner scan = new Scanner(System.in);
        System.out.print( “Enter a number: “ );
        int number = scan.nextInt();

        // print whether the number is odd or even
        if( number%2 == 0 )
            System.out.println( “even” );
        else System.out.println( “odd”);
   }
}
```

Complete Java Programs

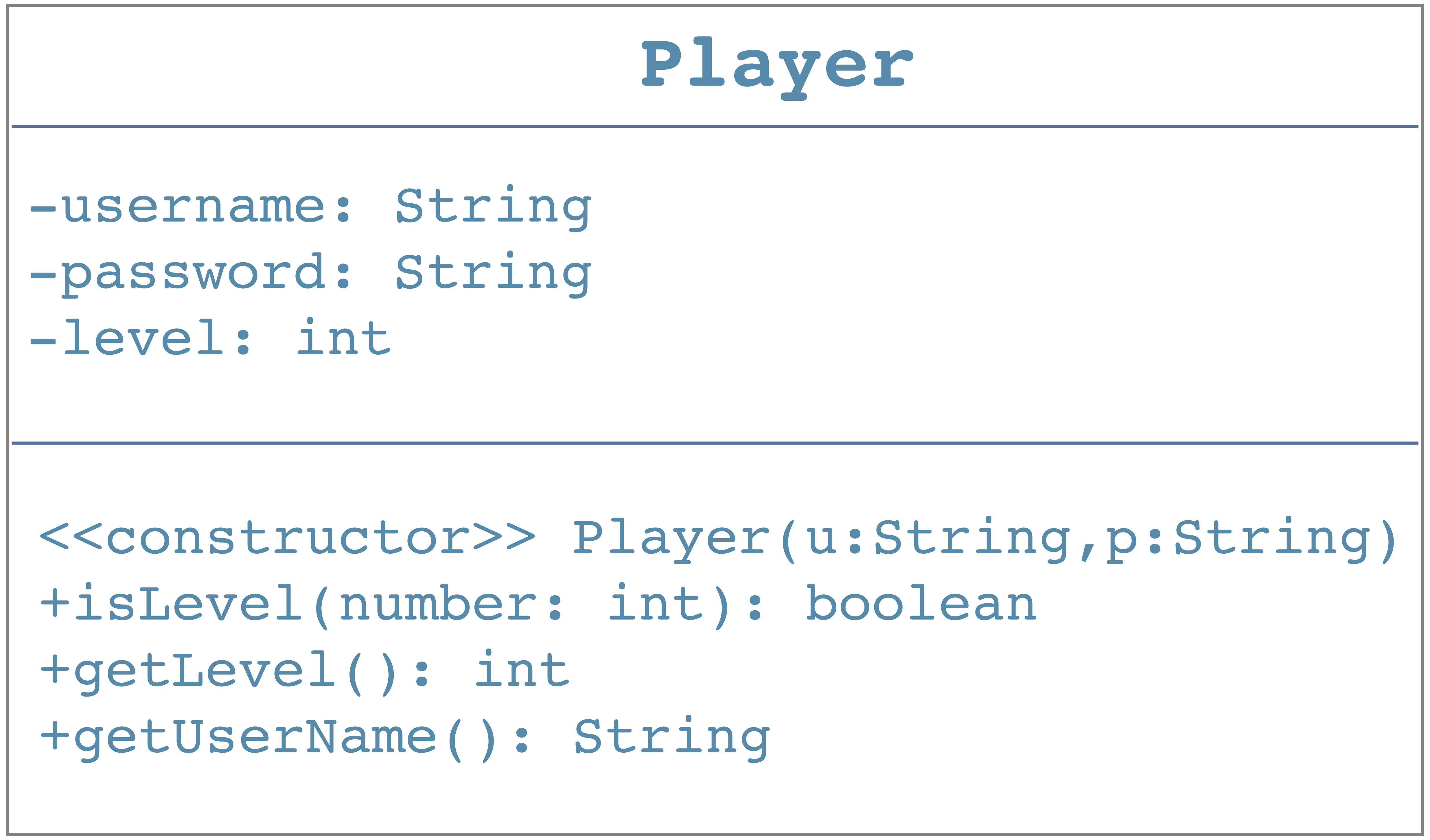
- Write the code for the Java class described in this UML diagram:



```
Player

-username: String
-password: String
-level: int

<<constructor>> Player(u:String,p:String)
+isLevel(number: int): boolean
+getLevel(): int
+getUserName(): String
```

Complete Java Programs

- Write the code for the Java class described in this UML diagram:

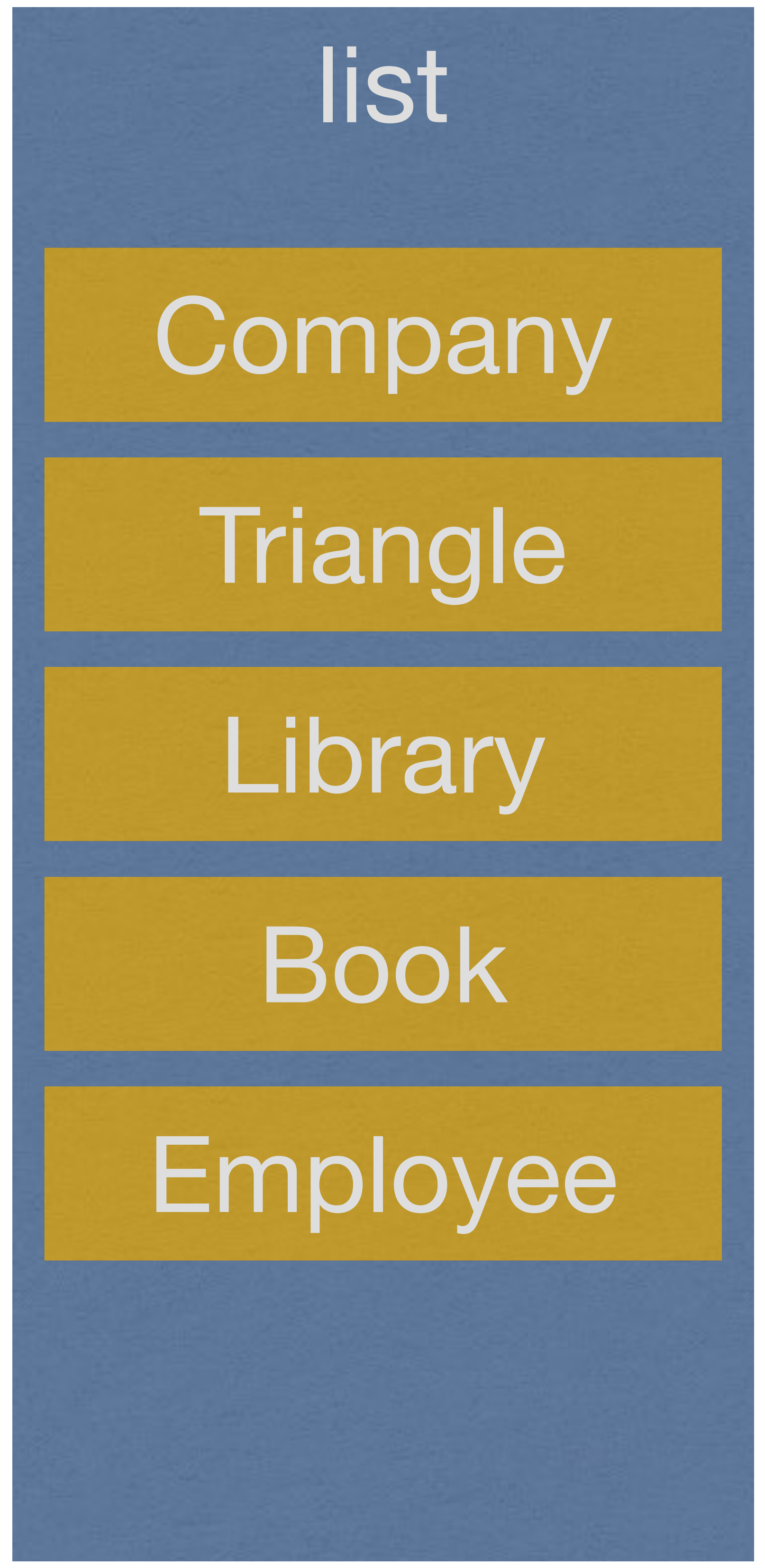


```
Player

-username: String
-password: String
-level: int

<<constructor>> Player(u:String,p:String)
+isLevel(number: int): boolean
+getLevel(): int
+getUserName(): String
```

```java
public class Player{

    private String username;
    private String password;
    private int level;

    public Player( String u, String p ){
       this.username = u;
       this.password = p;
     }

     public boolean isLevel( int number ){
        return number==this.level;
     }

     public int getLevel(){
        return this.level;
     }

     public String getUserName(){
        return this.username;
     }
}
```

What does the following code fragment print?

```
ArrayList<String> list = new ArrayList<String>();
list.add("Company");
list.add("Triangle");
list.add("Library");
list.add("Book");
list.add("Employee");
list.remove( list.size()-1 );

for( String val : list ) {
  System.out.print( val.charAt(3) );
}

System.out.println("");
System.out.println( list.size() );
```



UML

- Draw a UML diagram for the following classes:
  - The **Inventory** class has an ArrayList of **Items**.
  - Types of Items include basketballs, tennis racquets, and tennis balls.

What does the following code segment print?

```
int[] array = {1,2,3,2,1};
int v = 1;

for( int val : array ){
  if( v > val )
      v += val;
  else
      v -= val;

  System.out.printf(“%s”, val );
}

System.out.println(“”);
System.out.printf(“%d”, v );
```

What does the following code fragment print?

```
char[] letters = {‘A’, ‘B’, ‘2’, ‘L’, ‘U’, ‘H’, ‘K’, ‘m’, ‘G’, ‘c’, ‘Z’};
int[]  numbers = {3, 4, 9, 6};
String printMe = “”;

for( int index : numbers ){
    printMe += letters[index];
}

System.out.println( printMe + “!” );
```

What does the following main method print?

*given the code on the following 2 slides

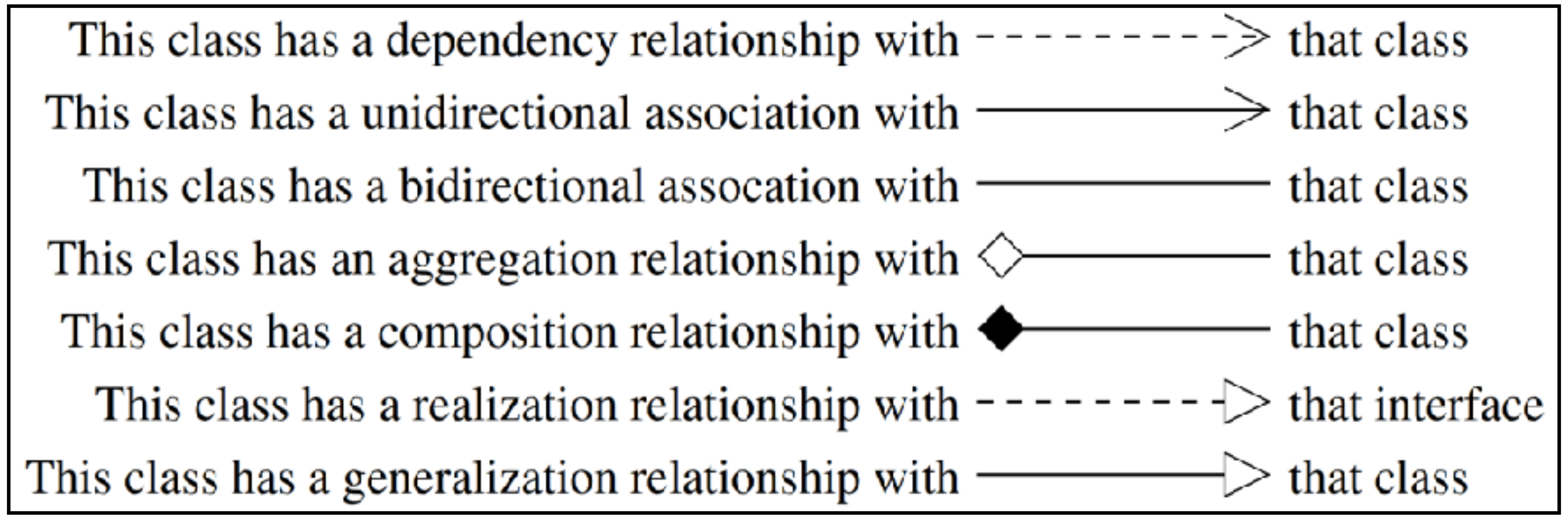
```
public class GeneratorTest {

  public static void main( String[] args ){

     Generator gen = new LetterGenerator("one","twenty",2);

    String t=" 3”;

     for( int i=0; i<gen.length(); i++ ){
        if( i==1 || i==2 || i==5 || i==7 )
            t = " 4";
        else t = " 3";
        System.out.println( gen.next() + t );
     }
  }

}
```

```
/**
 * A Generator supplies a series of values from a known list.
 * @author ASDF
 */
public interface Generator {
   /**
    * @return the next value generated
    */
   public char next();


   /**
    * @return the length
    */
   public int length();

}


/**
 * An AbstractGenerator gets the next value & returns value at an index.
 * @author ASDF
 */
public abstract class AbstractGenerator implements Generator{
    private int index;
    public abstract char getValueAt(int index);

    public char next(){
        index++;
        return getValueAt(index-1);
    }
}
```

```
/**
 * A LetterGenerator generates letters.
 * @author ASDF
 */
public class LetterGenerator extends AbstractGenerator {
  private String a,b,c;

  public LetterGenerator( String x, String y, int z ){
      this.a = x;
      this.b = y;
      this.c = String.valueOf( z );
  }

  public char getValueAt(int index){
      return this.a.charAt( index );
  }

  public int length(){
      return a.length();
  }
}
```

Draw a UML class diagram that shows the relationship between the classes in the previous question.

Recall that..



This class has a dependency relationship with	- - - - - - - ->	that class
This class has a unidirectional association with	———————>	that class
This class has a bidirectional assocation with	————————	that class
This class has an aggregation relationship with	◇———————	that class
This class has a composition relationship with	◆———————	that class
This class has a realization relationship with	- - - - - - - -▷	that interface
This class has a generalization relationship with	———————▷	that class

Suppose the first line in the GeneratorTest main method is changed

```
Generator gen = new NumberGenerator(4457.2, 135.7, 3043.2);
```

- Implement the NumberGenerator class so that the main method prints out the following:

```
4 3
4 4
5 4
7 3
. 3
2 4
```

- You may extend AbstractGenerator or directly implement Generator.

GeneratorTest question

- Does the code in the previous question adhere to the SOLID principles?

Question: LibraryTest

- Consider the main method in the `LibraryTest` class below for testing the `Book` class on this page and the `Library` class on the next page.

- Complete the `Library` class so that the main method of `LibraryTest` will print out:

```
The UTSA Library collection includes:
"Effective Java" by Joshua Bloch
"Sherlock Holmes" by Arthur Conan Doyle
"Sherlock Holmes" is available.
"Sherlock Holmes" is not available.
```

- All methods called in the main method must be implemented.

Question: LibraryTest

```
public class LibraryTest{

   public static void main( String[] args ){

       Library lib = new Library( “UTSA Library” );

       Book book1 = new Book(“Effective Java”, “Joshua Bloch”);
       Book book2 = new Book(“Sherlock Holmes”, “Arthur Conan Doyle”);
       lib.addBook( book1 );
       lib.addBook( book2 );

       Book book3 = new Book(“Sherlock Holmes”, “Arthur Conan Doyle”);
       boolean onShelf = lib.isAvailable( book3 );

       System.out.println( lib );
       System.out.println( “\”” + book3.getName() + “\””
              + (onShelf ? “is” : “is not”) + “ available” );

       lib.checkOut( book3 );
       onShelf = lib.isAvailable( book3 );
       System.out.println( “\”” + book3.getName() + “\””
              + (onShelf ? “is” : “is not”) + “ available” );
   }

}
```

Question: LibraryTest

```
public class Book {
  private String name;
  private String author;

  public Book( String n, String a ){
      this.name = n;
     this.author = a;
  }

  public String getName(){
      return this.name;
  }
}
```

Question: LibraryTest

```
import java.util.*;

public class Library {

   private String name;
   private ArrayList<Book> books;









   }
```