GEOG 178/258 -- WEEK 4:

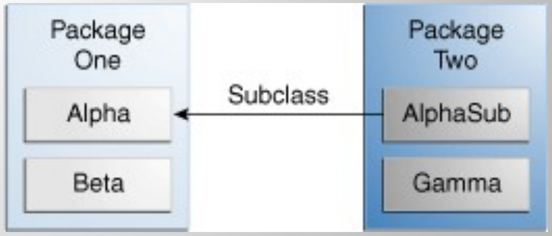
TO BE, OR NOT TO BE …

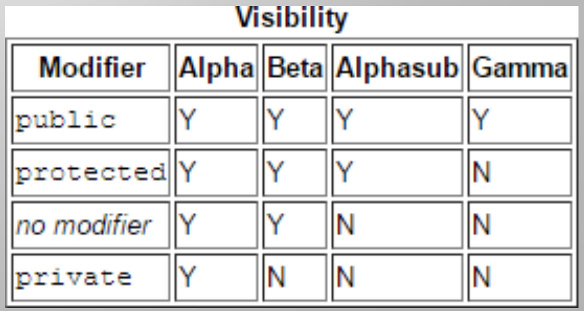
MIKE JOHNSON

PRIVATE VS. PUBLIC

Week 4

Private v. Public





Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

*Where are the member of the Alpha Class accessible

Tips on Choosing an Access Level:

If other programmers use your class, you want to ensure that errors from misuse cannot happen. Access levels can help you do this.

•Use the most restrictive access level that makes sense for a particular member. **Use private unless you have a good reason not to.**

• Avoid public fields except for constants. (Many of the examples in the tutorial use public fields. This may help to illustrate some points concisely, but is not recommended for production code.) Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.

TO BE OR NOT TO BE?

That is the question.

THE ‘FAMOUS’ QUESTION…

Week 4

Example

- The question “**To be or not to be….**” was first asked in Shakespeare’s Hamlet.
- How can we:
  - (A) model this question,
  - (B) model a human asking it, and
  - (C) model a debate of the question
- Lets assume that “**To be**” is the primary argument (TRUE) and “**Not to be**” is the secondary argument (FALSE).
- Lets also assume the question is fluid and every time it’s debated its state changes.
- We’ll also assume humans will change their mind to sync with reality after a debate
- Lets also assume that this question can only be asked so many times before it becomes pointless…

TO START

Week 4

Example

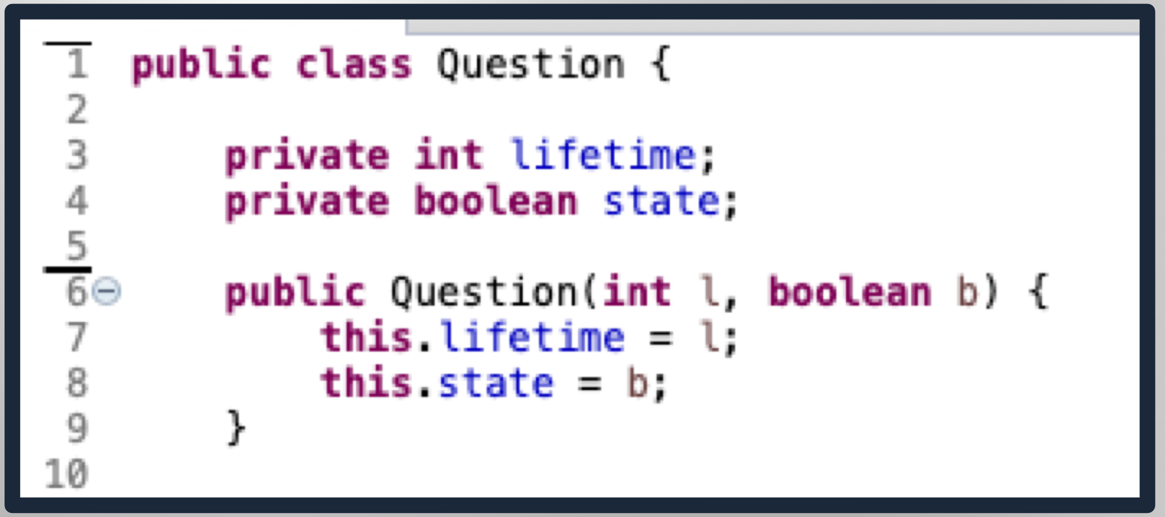
- **Create a new Java Project in a workspace under Week 4**
- **In this project file create three Java Classes**
  - 1) Question
  - 2) Human
  - 3) Test

1. QUESTION CLASS

Class and constructors

Week 4

Question Class



```
 1 public class Question {
 2 
 3     private int lifetime;
 4     private boolean state;
 5 
 6     public Question(int l, boolean b) {
 7         this.lifetime = l;
 8         this.state = b;
 9     }
10 
```

Line 1: Here we create the class **Question**

Line 3: We state that all questions have a **lifetime**. That is, a number of times it can be asked before it becomes useless. Here this value is declared as a **private** so it can not be set or changed outside of the 'blackbox' of the class Question.

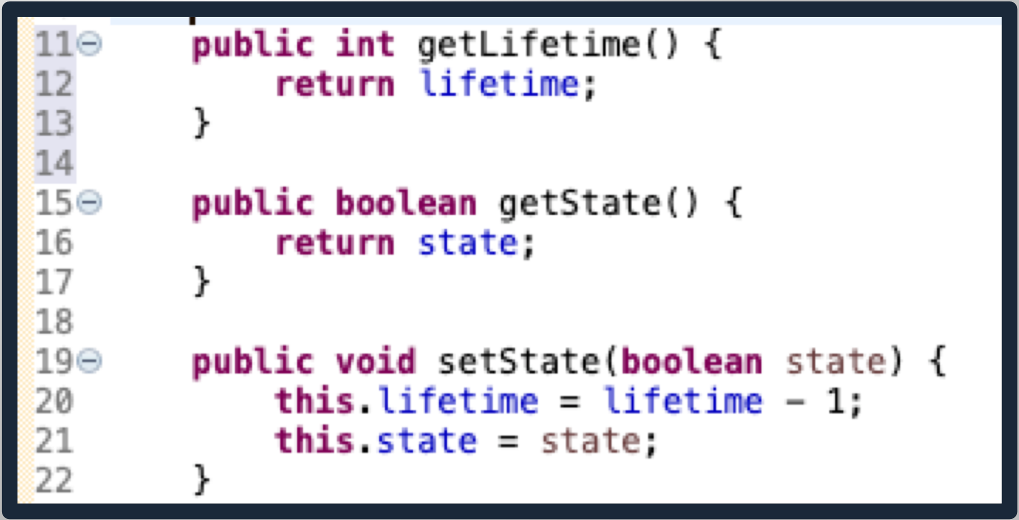
Line 4: All questions (in this model) also have a binary TRUE or FALSE condition. Again this variable is private as we don't want an outside entity to change the state of the question.

Line 6-9: CONSTRUCTOR here we require a user to provide a lifetime and state of the question.

Getters and setters

Week 4

Question Class



```
11     public int getLifetime() {
12         return lifetime;
13     }
14
15     public boolean getState() {
16         return state;
17     }
18
19     public void setState(boolean state) {
20         this.lifetime = lifetime - 1;
21         this.state = state;
22     }
```

Line 11 - 13: Here we create a getter method for lifetime (a public method that returns a integer)

Line 15 – 17: Here we create a getter method for state (a public method that returns a Boolean

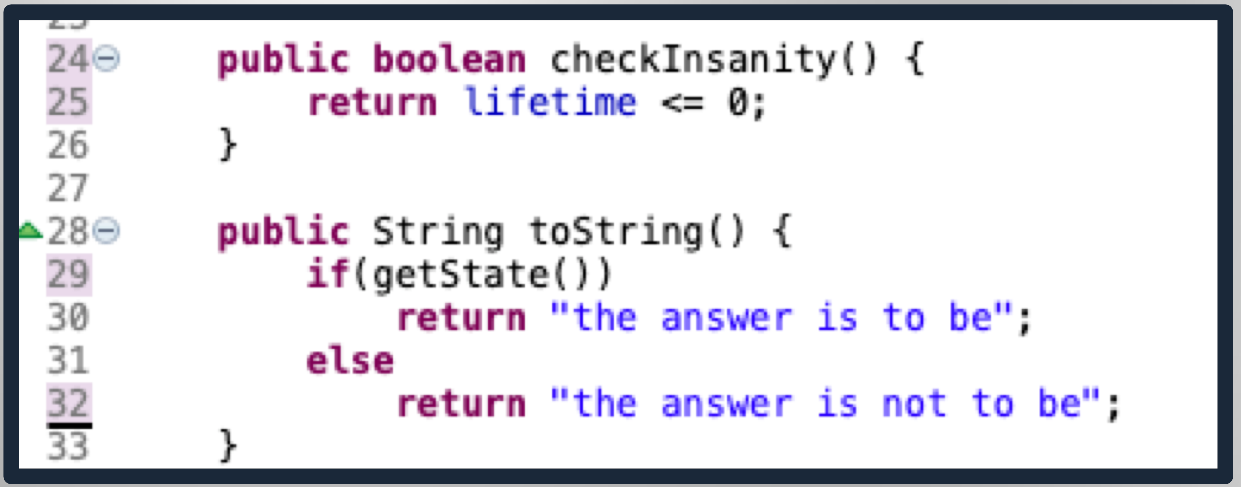
Line 19 – 22: Here we create a setter method for state(a public method that returns nothing and expects a Boolean value.

Why have we included the lifetime modifier within this method?

Define the methods

Week 4

Question Class



```
24      public boolean checkInsanity() {
25          return lifetime <= 0;
26      }
27
28      public String toString() {
29          if(getState())
30              return "the answer is to be";
31          else
32              return "the answer is not to be";
33      }
```

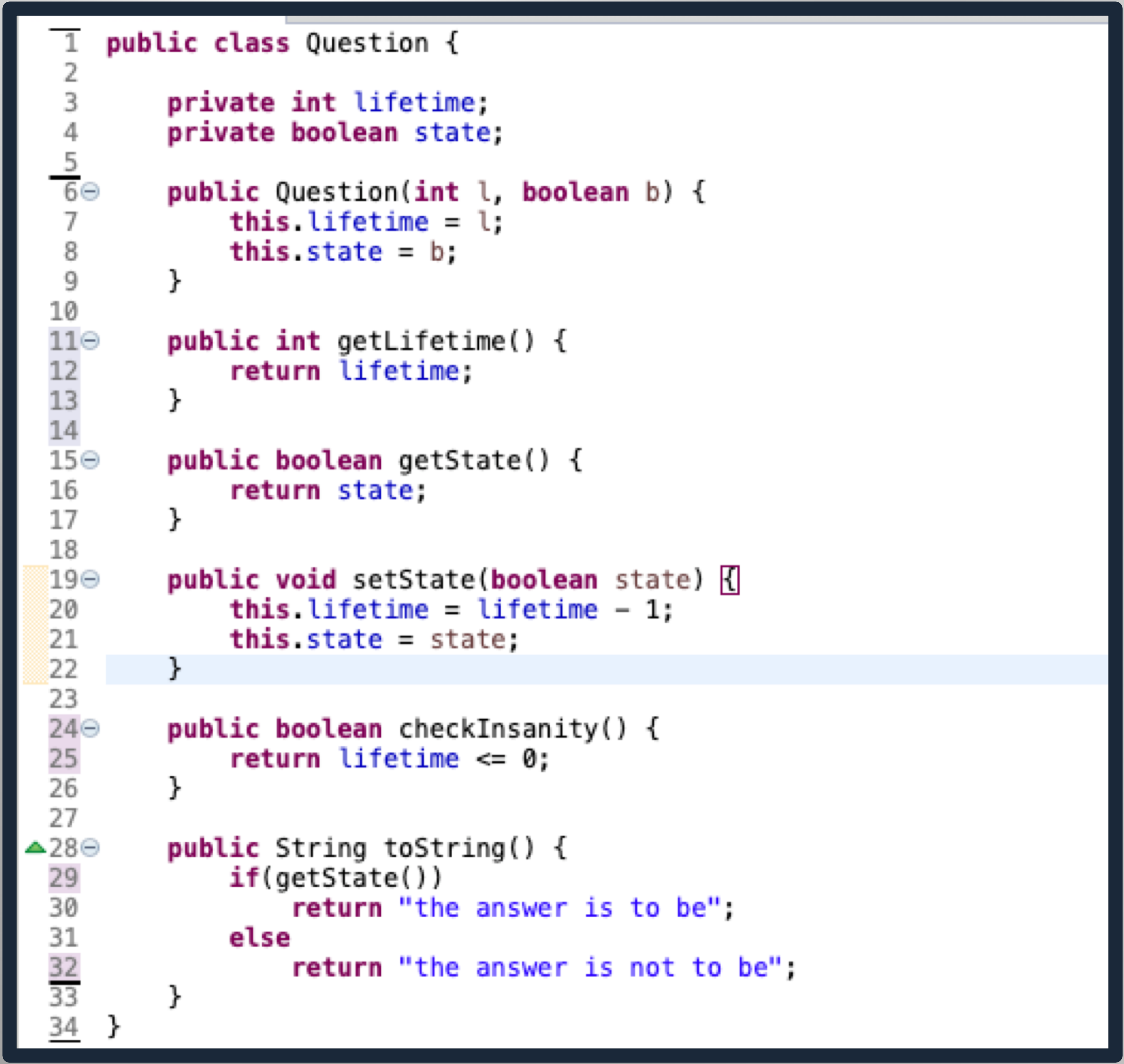
Line 24 - 26: Here we create a method that will return whether or not the lifetime of a question has dropped to zero (a public method that returns a Boolean value).

Line 28 – 32: Here we override the toString method telling a Question object how to interpret a Boolean state condition as a string. Note that we are calling on the getState getter to implement this method. (think back to assumption 1)

Week 4

Question Class

Complete code:



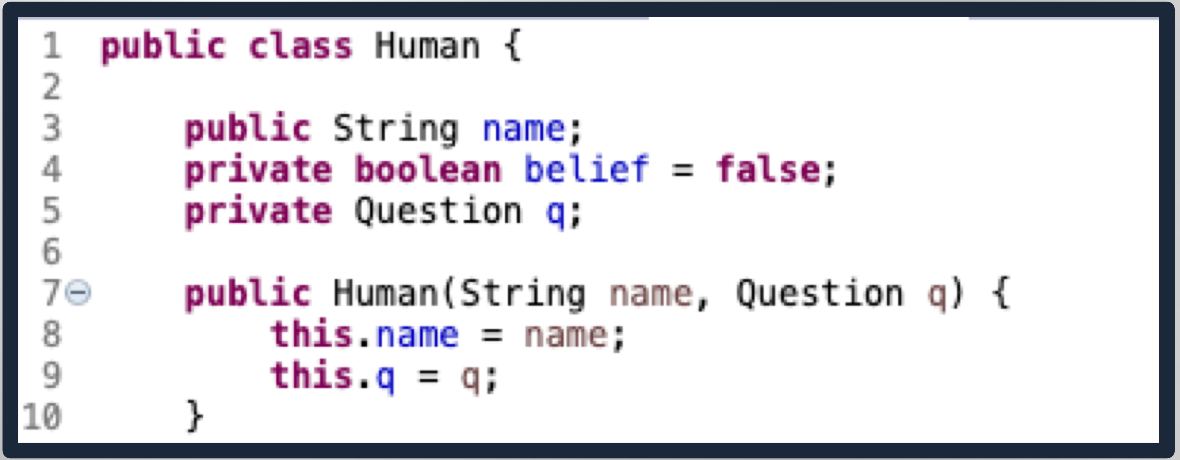
```java
public class Question {

    private int lifetime;
    private boolean state;

    public Question(int l, boolean b) {
        this.lifetime = l;
        this.state = b;
    }

    public int getLifetime() {
        return lifetime;
    }

    public boolean getState() {
        return state;
    }

    public void setState(boolean state) {
        this.lifetime = lifetime - 1;
        this.state = state;
    }

    public boolean checkInsanity() {
        return lifetime <= 0;
    }

    public String toString() {
        if(getState())
            return "the answer is to be";
        else
            return "the answer is not to be";
    }
}
```

2. HUMAN CLASS

Class and construction…

Week 4

Human Class



```
 1 public class Human {
 2
 3     public String name;
 4     private boolean belief = false;
 5     private Question q;
 6
 7     public Human(String name, Question q) {
 8         this.name = name;
 9         this.q = q;
10     }
```

Line 2: Create the **class** of Human making sure that it is public…

Line 3: Here we say that all humans have a name (String) that is publicly accessible

Line 4: All humans have a belief about a question. We assume that humans are always skeptical and will not believe a statement until its been debated and proven TRUE, and that their belief can only be altered by themselves (private)

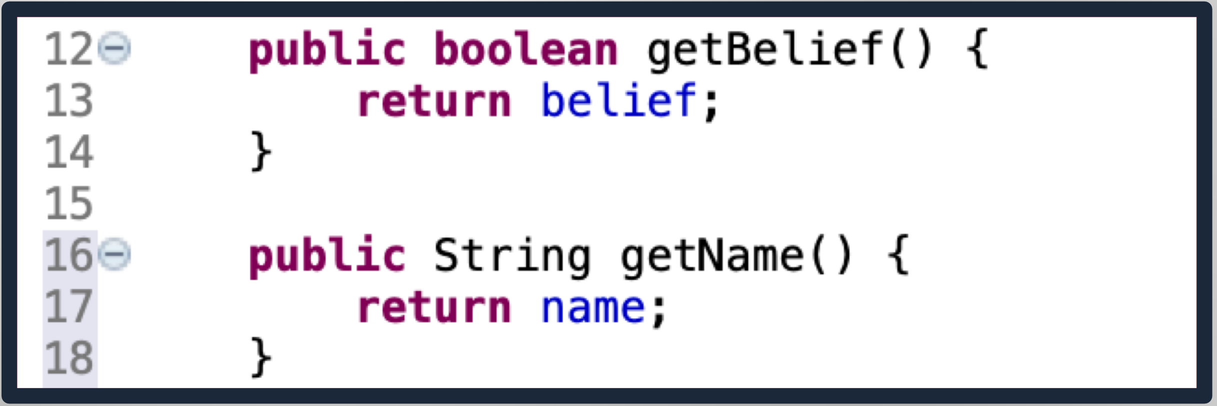
Line 5: In this model all humans exist with a question in mind, this question is part of their fundamental 'existence' in our data model.

Line 7 – 10: CONSTRUCTOR, all humans are created from an input name and question. These inputs become a central member of the specific human object.

Getters

Week 4

Human Class



```
12     public boolean getBelief() {
13         return belief;
14     }
15
16     public String getName() {
17         return name;
18     }
```

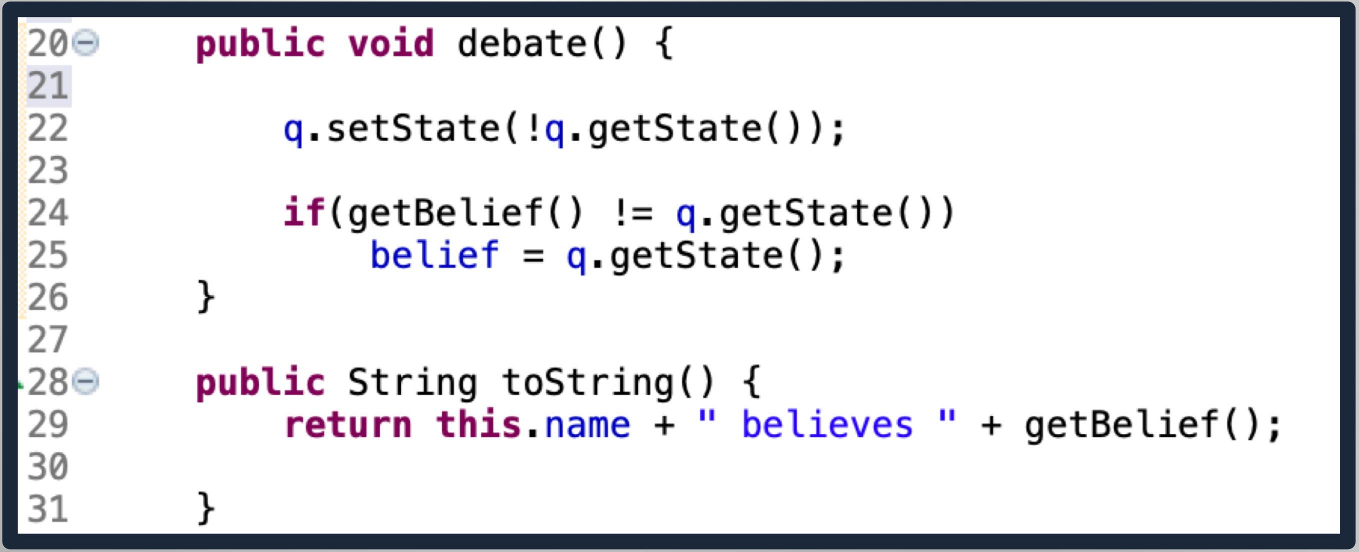
Line 12 - 14: Here we create a public get method from returning a human objects belief about the question.

Line 16 - 18: Here we create a public get method from returning a human objects name.

Methods

Week 4

Human Class



```
20      public void debate() {
21
22          q.setState(!q.getState());
23
24          if(getBelief() != q.getState())
25              belief = q.getState();
26      }
27
28      public String toString() {
29          return this.name + " believes " + getBelief();
30
31      }
```

Line 20 - 26: Here we are creating a public method called debate(). This method requires no inputs and returns a no values. In it, the function first sets the state of the current question to the opposite of its current state (respecting our assumption 2), and then aligns the current belief of the human with that reality (assumption 3)

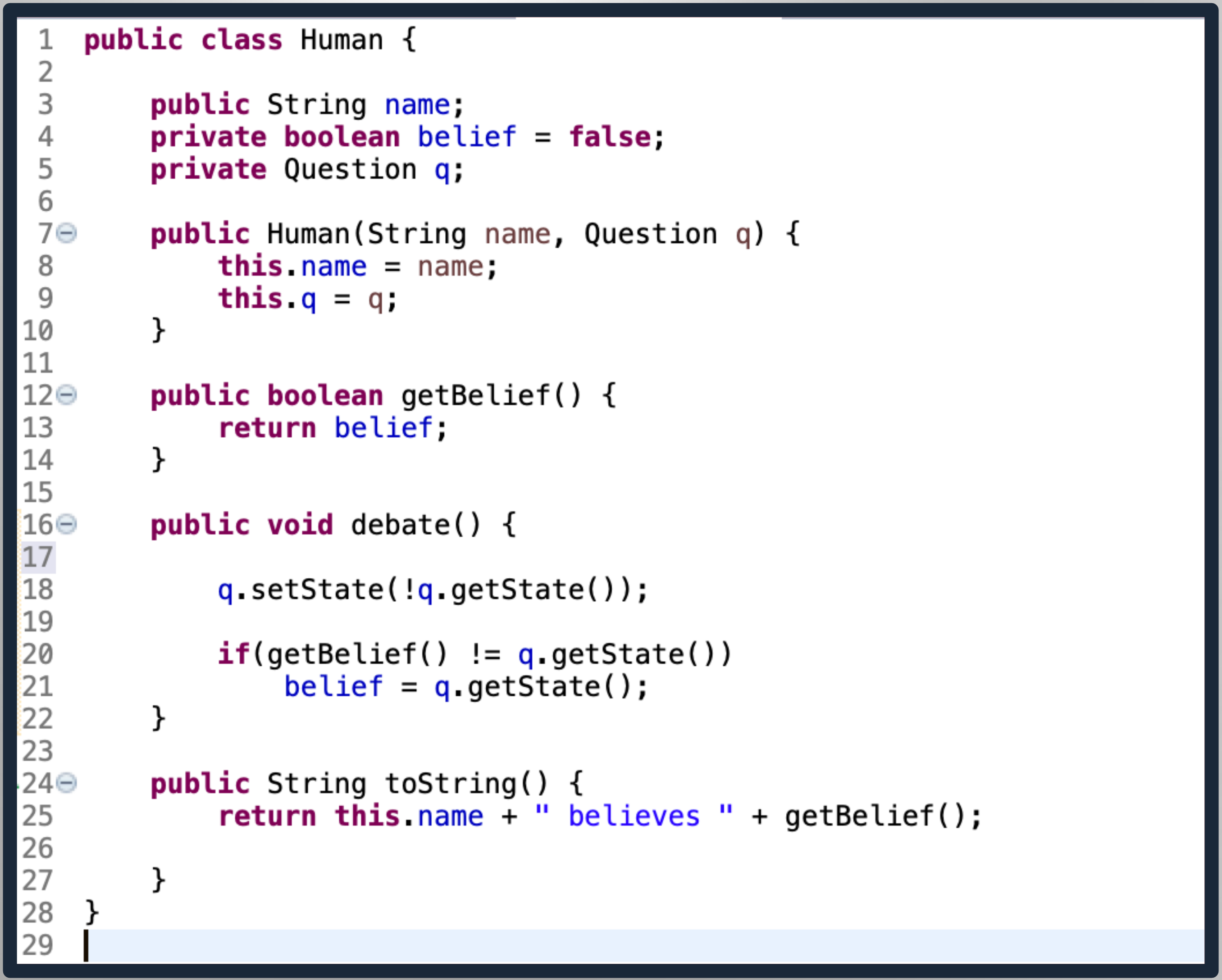
What happens if line 22 dnd 24:25 are switched??

Line 28 – 31: Here we over ride the toString method to return a more useful print statement with a public method.

Complete code…

Week 4

Human Class



```
public class Human {

    public String name;
    private boolean belief = false;
    private Question q;

    public Human(String name, Question q) {
        this.name = name;
        this.q = q;
    }

    public boolean getBelief() {
        return belief;
    }

    public void debate() {

        q.setState(!q.getState());

        if(getBelief() != q.getState())
            belief = q.getState();
    }

    public String toString() {
        return this.name + " believes " + getBelief();

    }
}
```

4. TEST CLASS

Week 4

Test Class

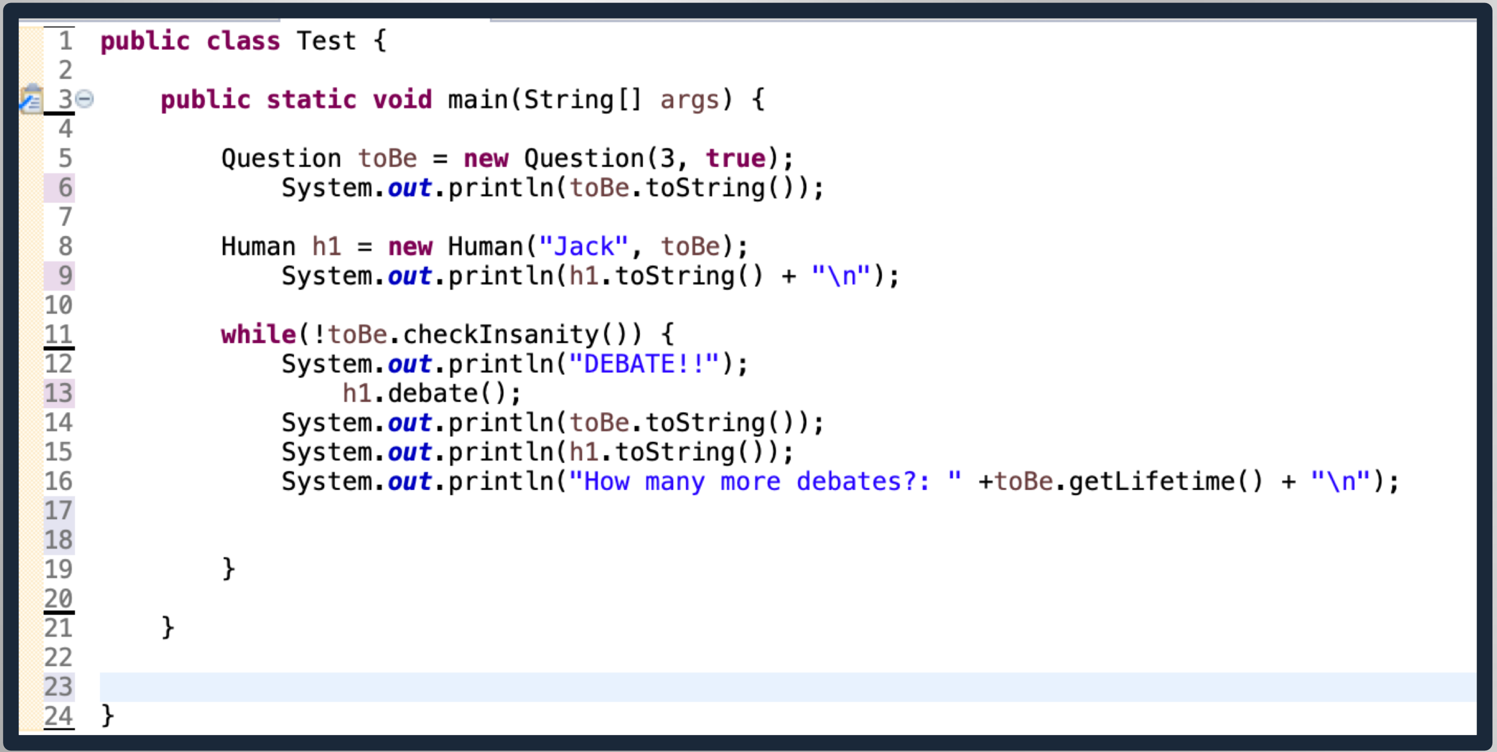
CHALLENGE!!

1. Can you combine these methods in the TEST class to ask/answer meaningful questions?
2. How can you combine methods to be sure that the human and the question are in sync?
3. Can you break your program?
   - How can you fix it?
4. What happens if a human gets new information and the lifespan of the question should be reset?
5. What other issues can you find? How might you fix them?

EXAMPLE…

Week 4

Test Class

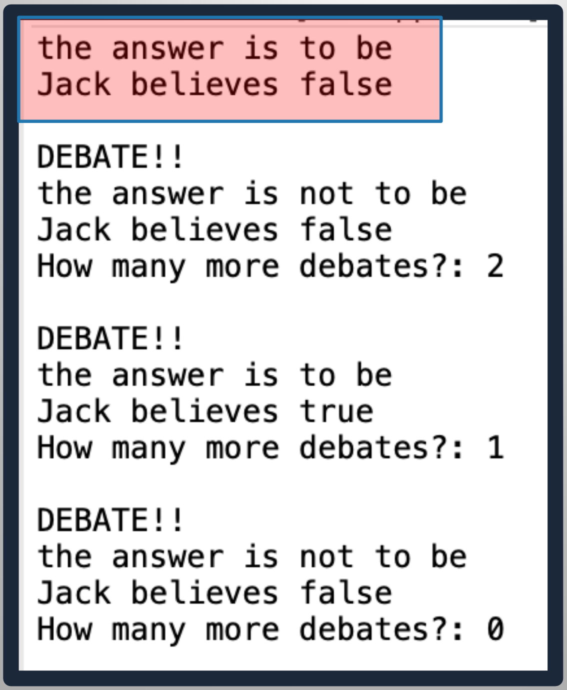


```
public class Test {

    public static void main(String[] args) {

        Question toBe = new Question(3, true);
            System.out.println(toBe.toString());

        Human h1 = new Human("Jack", toBe);
            System.out.println(h1.toString() + "\n");

        while(!toBe.checkInsanity()) {
            System.out.println("DEBATE!!");
                h1.debate();
            System.out.println(toBe.toString());
            System.out.println(h1.toString());
            System.out.println("How many more debates?: " +toBe.getLifetime() + "\n");


        }

    }


}
```

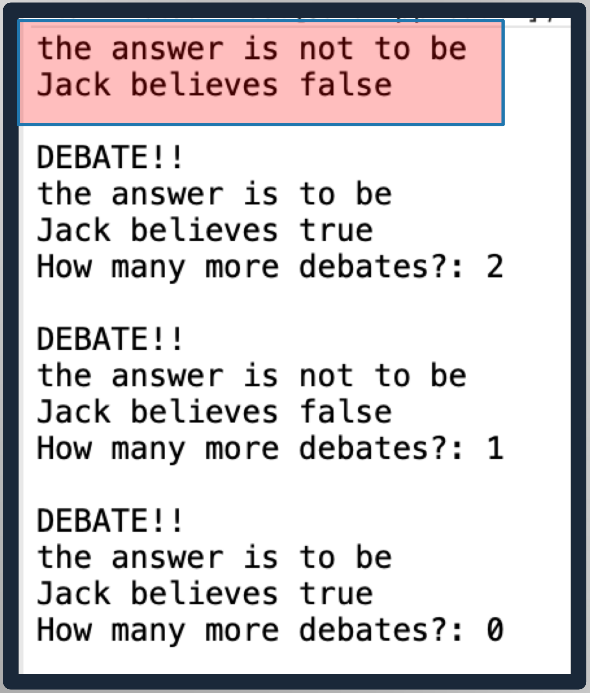
Output!

Week 4

Test Class



```
the answer is to be
Jack believes false

DEBATE!!
the answer is not to be
Jack believes false
How many more debates?: 2

DEBATE!!
the answer is to be
Jack believes true
How many more debates?: 1

DEBATE!!
the answer is not to be
Jack believes false
How many more debates?: 0
```



```
the answer is not to be
Jack believes false

DEBATE!!
the answer is to be
Jack believes true
How many more debates?: 2

DEBATE!!
the answer is not to be
Jack believes false
How many more debates?: 1

DEBATE!!
the answer is to be
Jack believes true
How many more debates?: 0
```

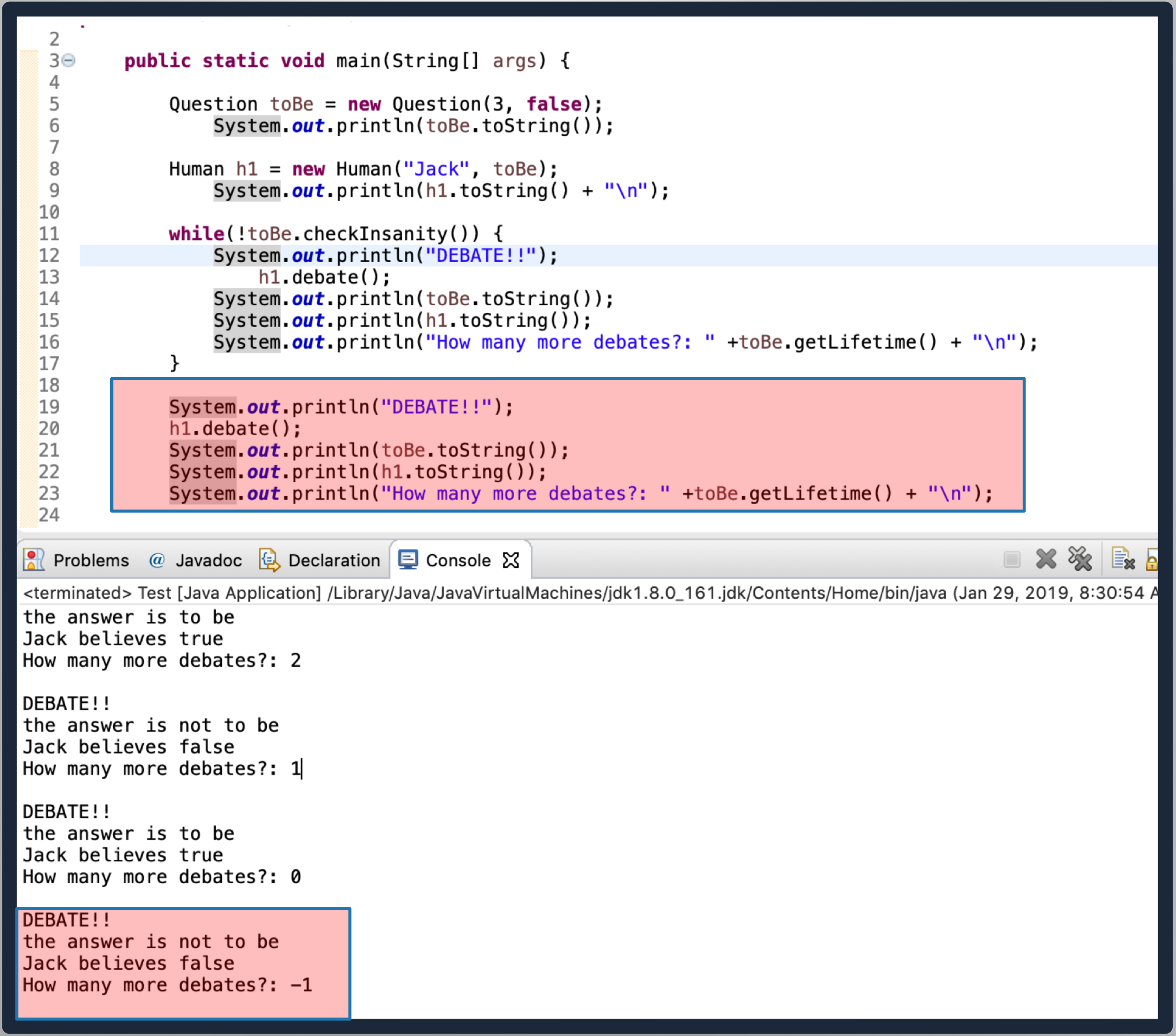
This code is not fully complete… can you find a reason why?

ANSWER…

Week 4

Test Class

In the codes current set up every time that Jack debates, even if the conversation has surpassed its lifespan, he will keep changing his mind…



```
	public static void main(String[] args) {

		Question toBe = new Question(3, false);
			System.out.println(toBe.toString());

		Human h1 = new Human("Jack", toBe);
			System.out.println(h1.toString() + "\n");

		while(!toBe.checkInsanity()) {
			System.out.println("DEBATE!!");
				h1.debate();
			System.out.println(toBe.toString());
			System.out.println(h1.toString());
			System.out.println("How many more debates?: " +toBe.getLifetime() + "\n");
		}

		System.out.println("DEBATE!!");
		h1.debate();
		System.out.println(toBe.toString());
		System.out.println(h1.toString());
		System.out.println("How many more debates?: " +toBe.getLifetime() + "\n");
```

```
<terminated> Test [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java (Jan 29, 2019, 8:30:54 A
the answer is to be
Jack believes true
How many more debates?: 2

DEBATE!!
the answer is not to be
Jack believes false
How many more debates?: 1

DEBATE!!
the answer is to be
Jack believes true
How many more debates?: 0

DEBATE!!
the answer is not to be
Jack believes false
How many more debates?: -1
```

SUMMARY…

Week 4

End

1. When creating classes it is important to think of what kind of questions and what kind of information you are interested in regarding an object.…
2. Think about how much of that information should be controlled by the user, and how much should be hidden…
3. Finally remember that all coding is “a work in progress” Unlike an essay, you don’t start at the beginning and write to the end.

Homework hints…

How are Question and a light bulb similar?
How are they different?

How are Human and light switch similar?
How are they different?

- **LightBulb**
  - public boolean isBroken()
  - public boolean isState()
  - public int getLifetime()

- **LightSwitch**
  - public boolean isState()
  - public void useSwitch()
  - public void replaceLightBulb(LightBulb lb)