CS 520

Theory and Practice of Software Engineering
Spring 2021

Collaborative development

March 4, 2021

Tuesday

- Requirements Engineering
  - Stakeholders
  - Software requirements specification
  - 4 main phases: Elicitation, Specification, Analysis, Management
- Specification
  - Non-functional (commonly called the “ilities”)
  - Functional: Natural Language, Finite State Automata (FSAs), Property Pattern Specifications, Propel

Example: Online banking app

Your project is to develop an online banking app for U.S. customers:

- Supports individual and corporate customers
- Runs on desktops, tablets, and cellphones
- Provides checking and savings accounts
- Each account should support the following transactions: deposit, withdrawal, transfer.
- ...

Online banking app: Stakeholders

- Customers: individual and corporate
- Bank: tellers, upper management, online help desk
- Financial companies (such as investment firms or credit cards) that want to be linked to customer accounts
- app company: developers (for implementation and testing), beta testers, upper management
- app store company (e.g., Google Play)
- Cloud service or database service
- UI experts: human factors and/or human-computer interaction (HCI)
- Federal agencies: ADA, FDIC, IRS

Online banking app:
Non-functional requirements (“ilities”)

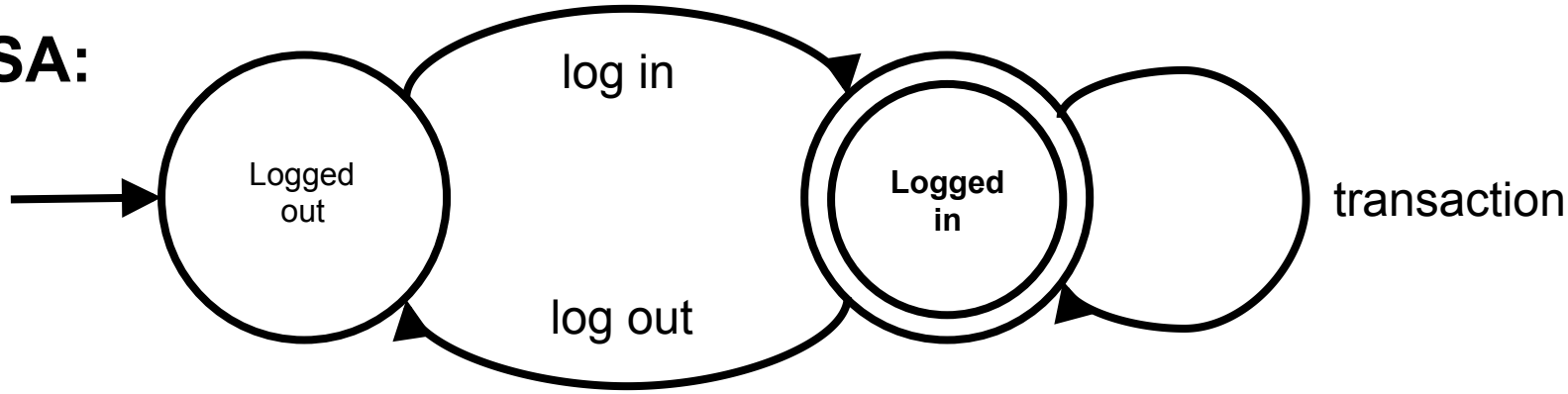
- Scalability
- Reliability
- Availability
- Usability
- Testability
- Extendibility
- Vulnerability
- Supportability
- Security
- Privacy

https://en.wikipedia.org/wiki/Non-functional_requirement

Online banking app: Functional requirements

- **Natural language:** After a banking customer logs in to this app, they can perform transactions on their accounts until they log out of the app

- **FSA:**



Property specification pattern:

- *Scope: Between log in and log out*
- *Behavior: Existence of transaction*

Today

- Agile development
- Scrum
- Pair programming
- Collaborative development exercise
- Final project selection

Agile development

- Fast paced
- Frequent releases
- Developer centered
  - Do we need managers?

Scrum

- A very popular flavor of Agile to rapidly iterate in Sprints
  - Each Sprint develops then releases the product
- Three pillars:
  - Transparency
  - Inspection
  - Adaptation
- Used by large tech companies such as Facebook, Google, Microsoft

https://www.scrum.org

Three roles

- Product owner
  - represents the customer specifying the goal
- Development team
  - Performs Sprints
  - delivers software product that satisfies that goal
- Scrum master
  - buffer between team and outside world
  - prevents distractions, barriers

Many aspects of Scrum

- Sprints
- Stand-up meetings
  - What did I do yesterday?
  - What will I do today?
  - Do I see any impediment from our goal?
- Reviews

Pair programming

- Requirements specification, designing, implementing, testing, etc.
- Pair-work facilitates
  - transparency
  - no single point of failure
  - decision making
  - focus
  - creativity

Collaborative development exercise

- Further develop a Figure editor available here: https://github.com/LASER-UMASS/cs520-Spring2020.git

- Form pairs that will collaboratively work on specification, design, and implementation

Figure editor

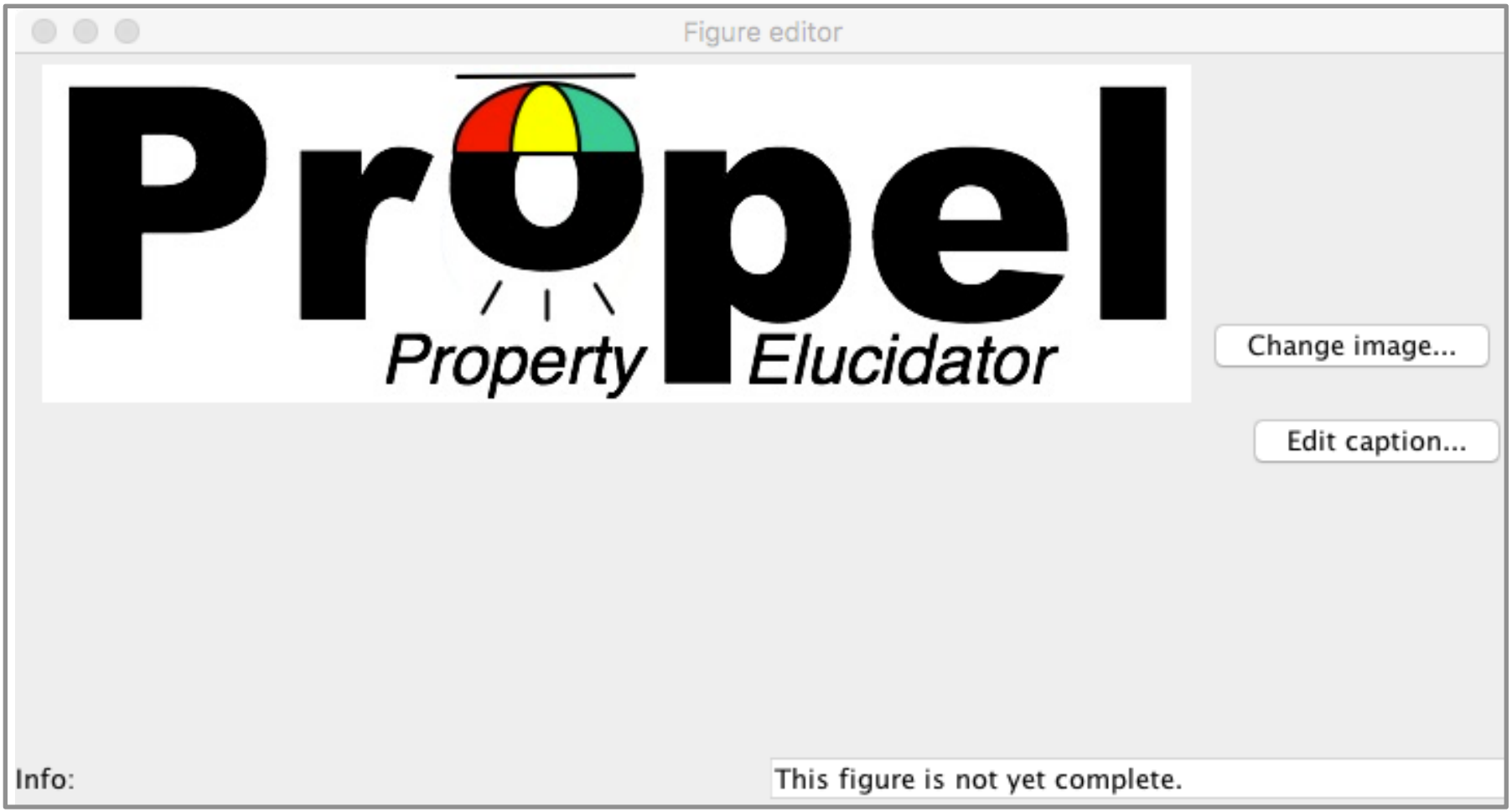
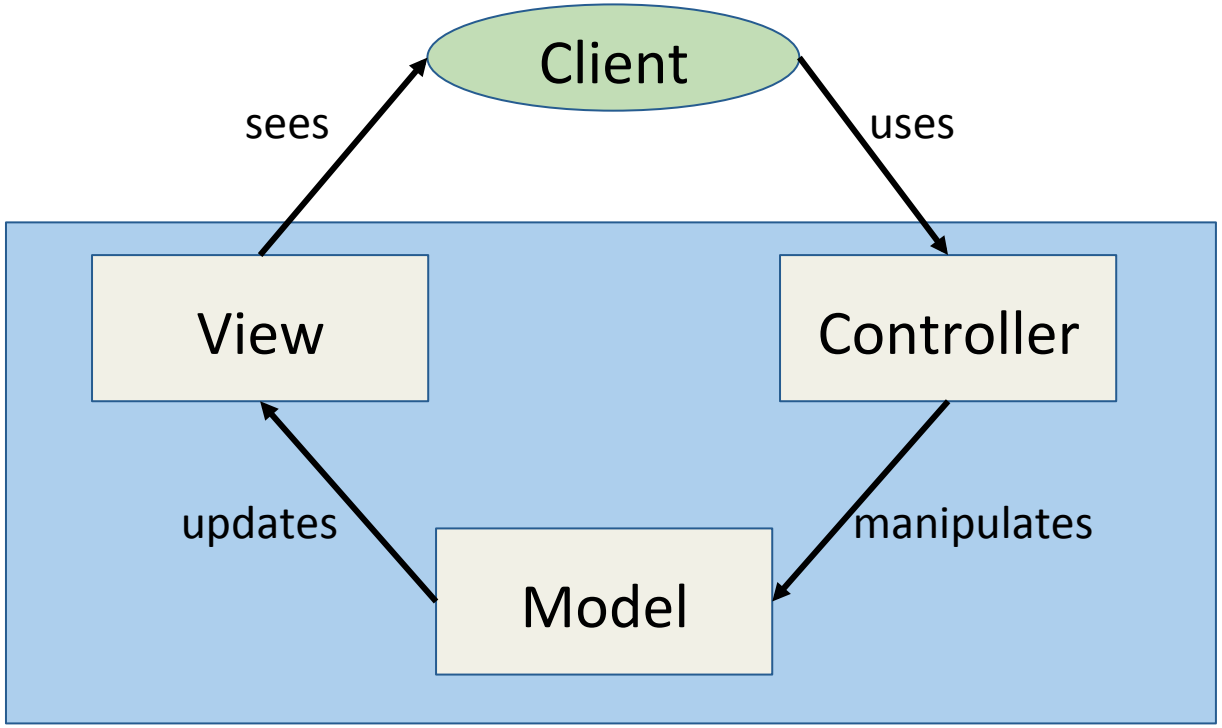


Figure editor (v1): MVC architecture



Separates data representation (Model),
visualization (View), and client interaction (Controller)

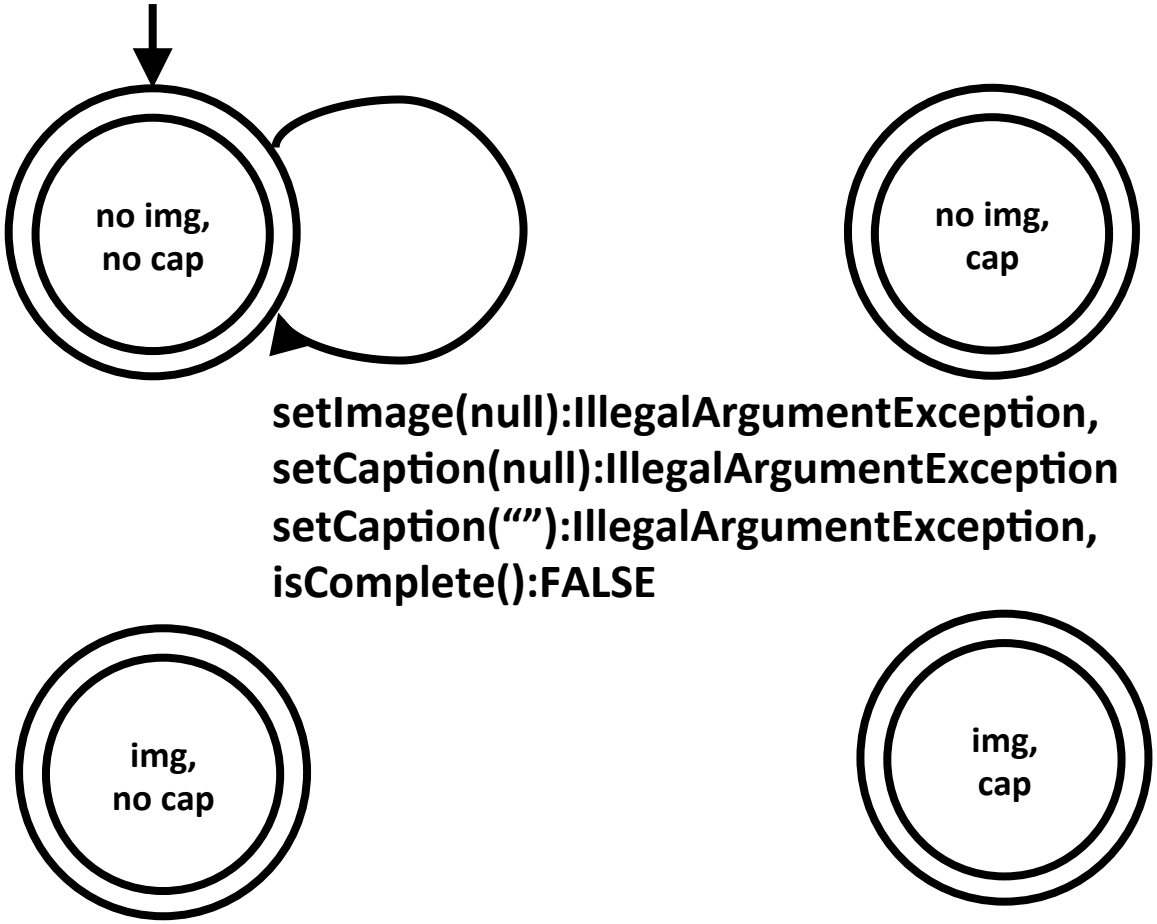
Figure editor (v1): Model API

All Methods | Instance Methods | Concrete Methods

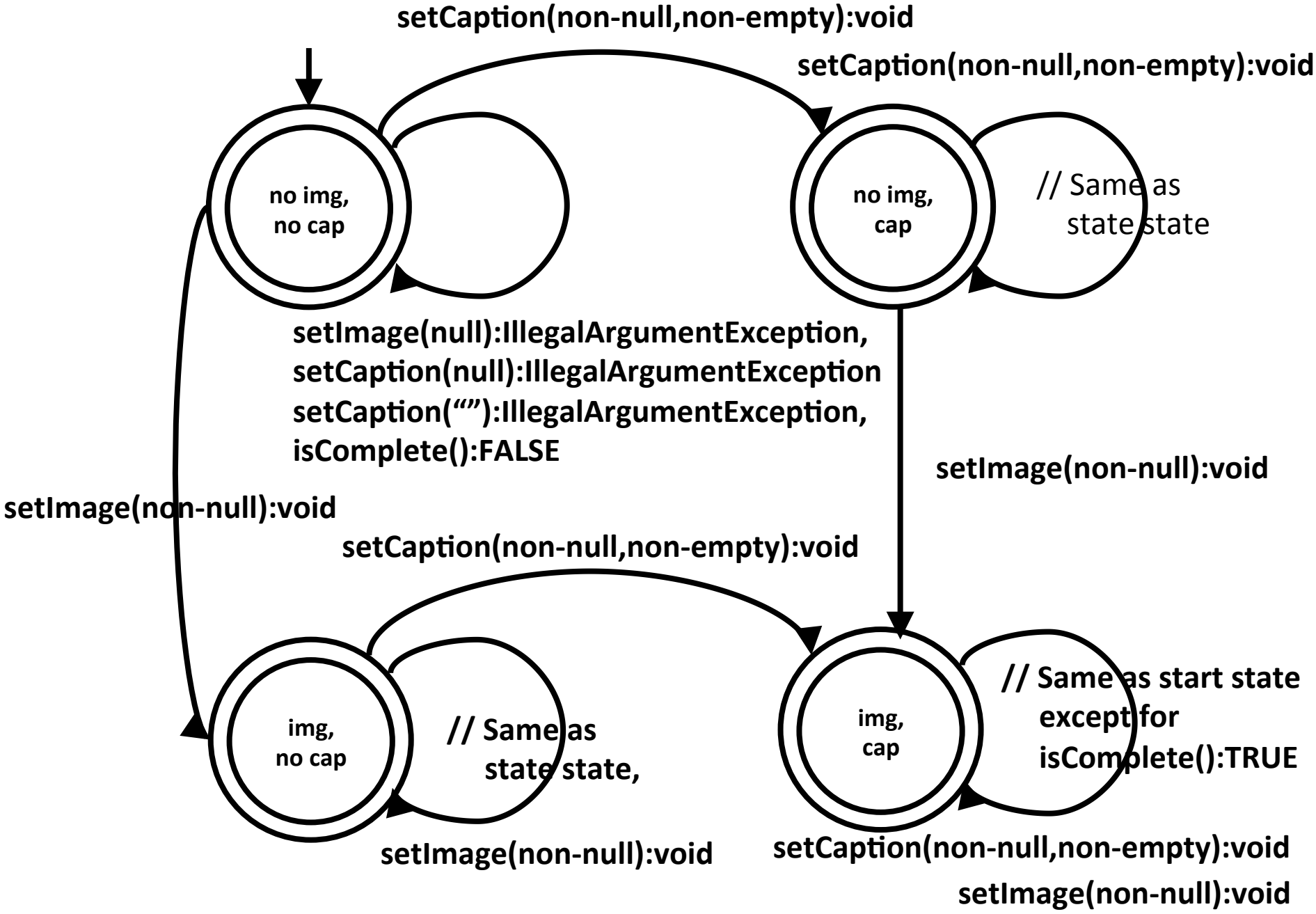
Modifier and Type	Method and Description
`java.lang.String`	**`getCaption()`**
`javax.swing.ImageIcon`	**`getImage()`**
`boolean`	**`isComplete()`** Returns true if this figure is complete, meaning its Image is non-null and its caption is non-null and non-empty, and false otherwise.
`void`	**`setCaption(java.lang.String newCaption)`** Sets the caption to the given non-null and non-empty String.
`void`	**`setImage(javax.swing.ImageIcon newImage)`** Sets the image to the given non-null ImageIcon.

Model (v1): FSA specification

- Complete the behavioral specification (written as an FSA)



NOTE) The violation state is implied (not shown).


setCaption(non-null,non-empty):void
setCaption(non-null,non-empty):void
no img,
no cap
no img,
cap
// Same as
state state
setImage(null):IllegalArgumentException,
setCaption(null):IllegalArgumentException
setCaption(""):IllegalArgumentException,
isComplete():FALSE
setImage(non-null):void
setImage(non-null):void
setCaption(non-null,non-empty):void
img,
no cap
// Same as
state state,
img,
cap
// Same as start state
except for
isComplete():TRUE
setImage(non-null):void
setCaption(non-null,non-empty):void
setImage(non-null):void

Model (v1): Implementation

- Use the Model API and FSA for the implementation

setImage

```
public void setImage(javax.swing.ImageIcon newImage)
```

Sets the image to the given non-null ImageIcon.

Parameters:

`newImage` - The ImageIcon must be non-null

Throws:

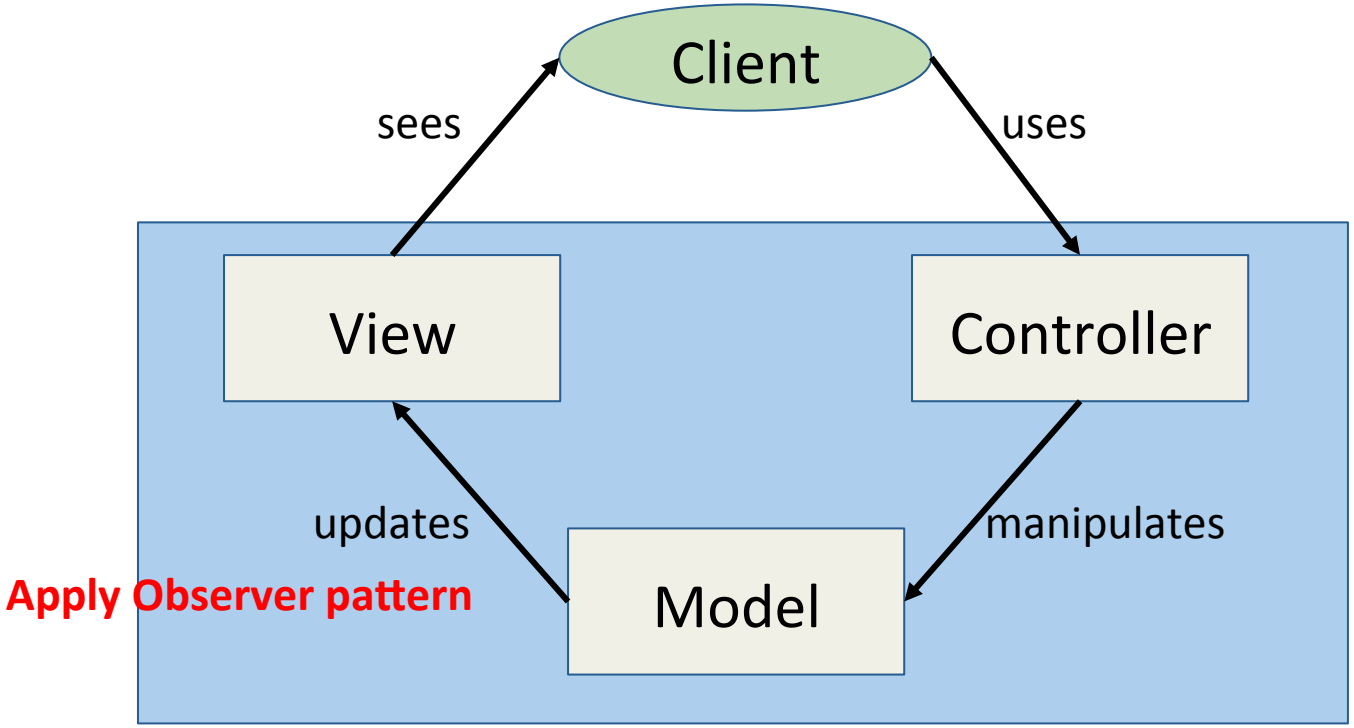
`java.lang.IllegalArgumentException` - if the ImageIcon is null

```
public class FigureModel
{
   private ImageIcon image; // Support information hiding (encapsulation) with visibility)
   private String caption; // For readability/understandability, use words not single chars

   public ImageIcon getImage() { return this.image; }

   public void setImage(ImageIcon newImage) {
      // Check the pre-condition (i.e. input validation)
      if (newImage == null) { throw new IllegalArgumentException("Image must be non-null."); }
      this.image = newImage;
   }

   public String getCaption() { return this.caption; }

   public void setCaption(String newCaption) {
      if ((newCaption == null) || newCaption.trim().equals(""))
      { throw new IllegalArgumentException(…); }
      this.caption = newCaption;
    }

    public boolean isComplete() {
       return ((this.image != null) && (this.caption != null) && (! this.caption.equals("")));
    }
}
```

```
public class FigureModel
{
   private ImageIcon image; // Support information hiding (encapsulation) with visibility)
   private String caption; // For readability/understandability, use words not single chars

   public ImageIcon getImage() { return this.image; }

   public void setImage(ImageIcon newImage) {
      // Check the pre-condition (i.e. input validation)
      if (newImage == null) { throw new IllegalArgumen                    ); }
      this.image = newImage;
   }

   public String getCaption() { return this.caption; }

   public void setCaption(String newCaption) {
      if ((newCaption == null) || newCaption.trim().equals(“”))
      { throw new IllegalArgumentException(…); }
      this.caption = newCaption;
    }

    public boolean isComplete() {
       return ((this.image != null) && (this.caption != null) && (! this.caption.equals(“”)));
    }
}
```

This class actually has javadoc comments that were used to generate the hypertext description shown earlier in the slides.

Figure editor (v2): MVC architecture



Separates data representation (Model),
visualization (View), and client interaction (Controller)

Figure editor (v1): Observer pattern

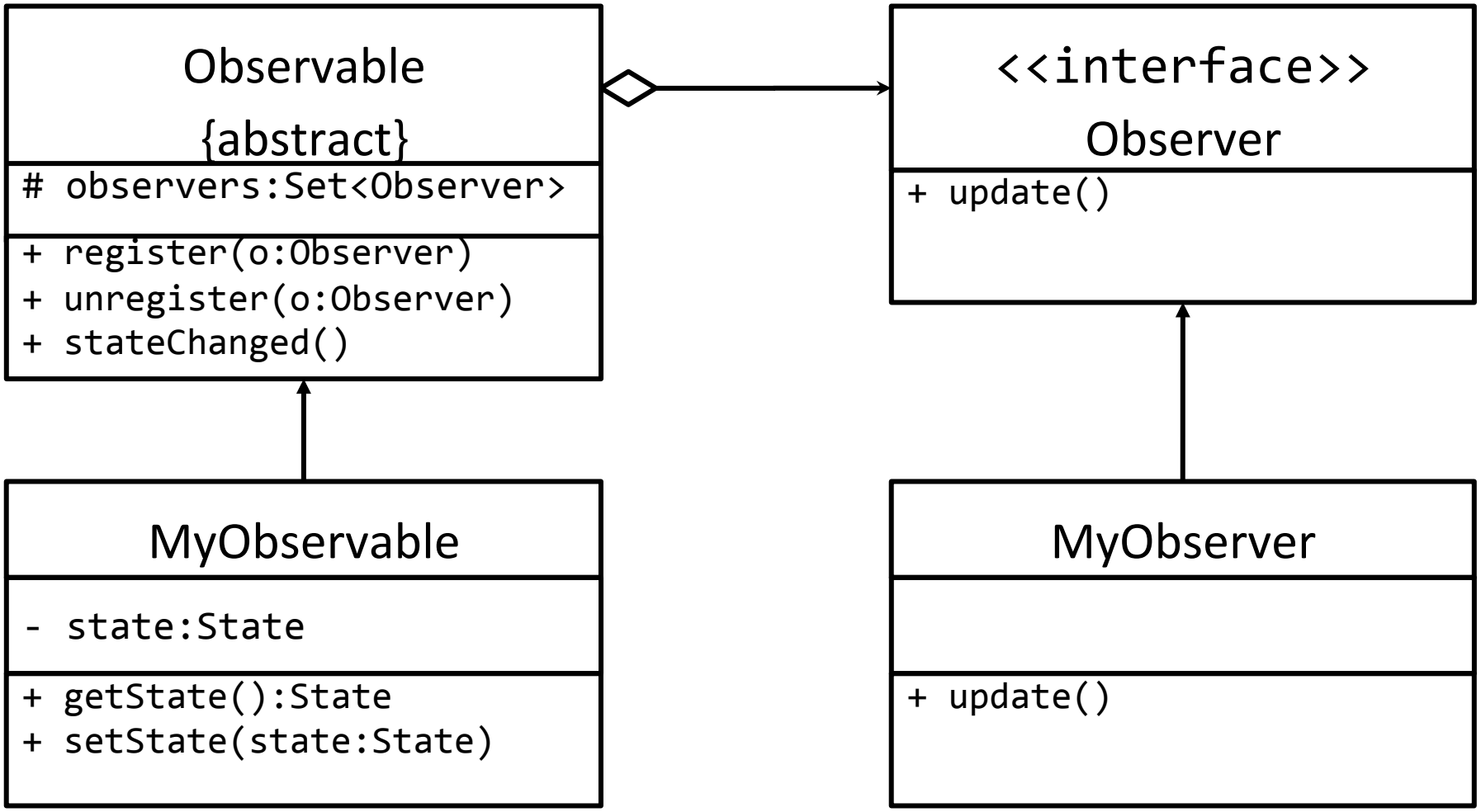


Figure editor (v1): Observer pattern

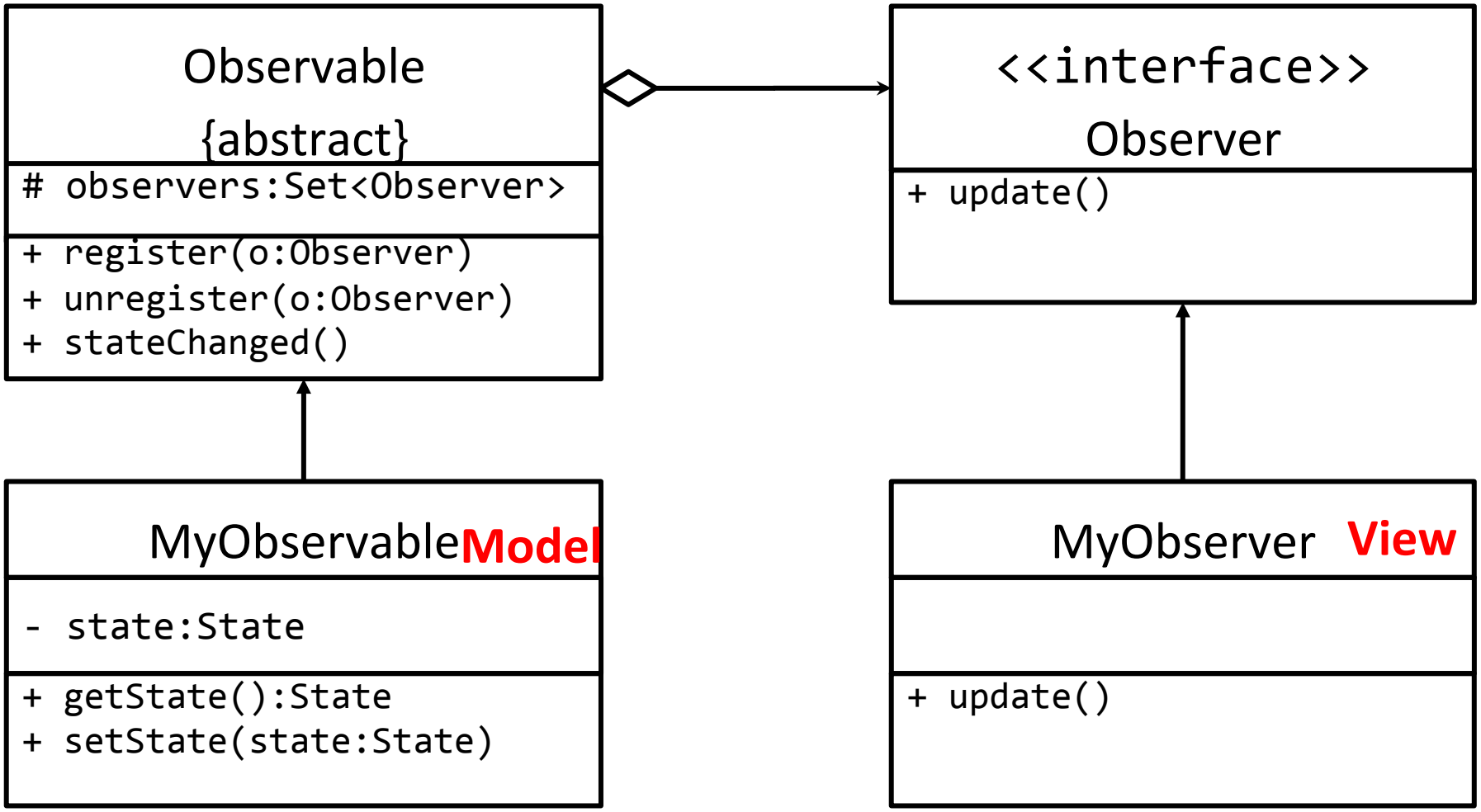


Figure editor (v2): Implementation

Java provides the following two classes:

- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html
- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html

How could the Observer design pattern be implemented using these classes?

Figure editor (v2): Implementation

Java provides the following two classes:

- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html
- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html

How could the Observer design pattern be implemented using these classes?

- **Observable:** FigureModel has-a PropertyChangeSupport
- **Observer:** Each View is-a PropertyChangeListener

Figure editor (v2): Implementation

Java provides the following two classes:

- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeSupport.html
- https://docs.oracle.com/en/java/javase/15/docs/api/java.desktop/java/beans/PropertyChangeListener.html

How could the Observer design pattern be implemented using these classes?

- **Observable:** FigureModel has-a PropertyChangeSupport
- **Observer:** Each View is-a PropertyChangeListener

NOTE) The Week 5 Participation Questionnaire will discuss this further.

Topics covered

- Documentation, e.g.,
  - README, javadoc, internal comments
- Specification, e.g.,
  - Natural language, FSAs
- Architecture & design, e.g.,
  - Patterns (MVC, Observer)
  - Class diagrams
- Implementation
  - Pair programming
  - Java, Swing

Final project selection

- Form team of 4 or 5 students
- Select one of the following topics:
  1. MSR mining challenge: 2020 or 2021
  2. Replication study
  3. ML (Machine Learning) development toolkit
  4. EleNa: Elevation-based navigation
  5. Propose your own group project
- Due: Thursday March 4, 2021 9:00 PM

https://people.cs.umass.edu/~hconboy/class/2021Spring/CS520/finalProject.pdf

Final project: Selected topic

1. Read some background material
2. Start to develop
3. Create and give a mid-point presentation
4. Continue to develop
5. Create and give a final presentation
6. Document the final project (either a writeup for the first 3 topics or a version control repository for the 4th topic)

Final group logistics

- Choose a team leader or Agile?
- Hold weekly meetings or share weekly progress reports?
- Discuss experience with different languages and tools?
- Version control system (e.g., git, Google Drive)?