COMP 504: Graduate Object-Oriented Programming and Design

Lecture 7: Use Cases and Strategy Design Pattern

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp504

Use Cases

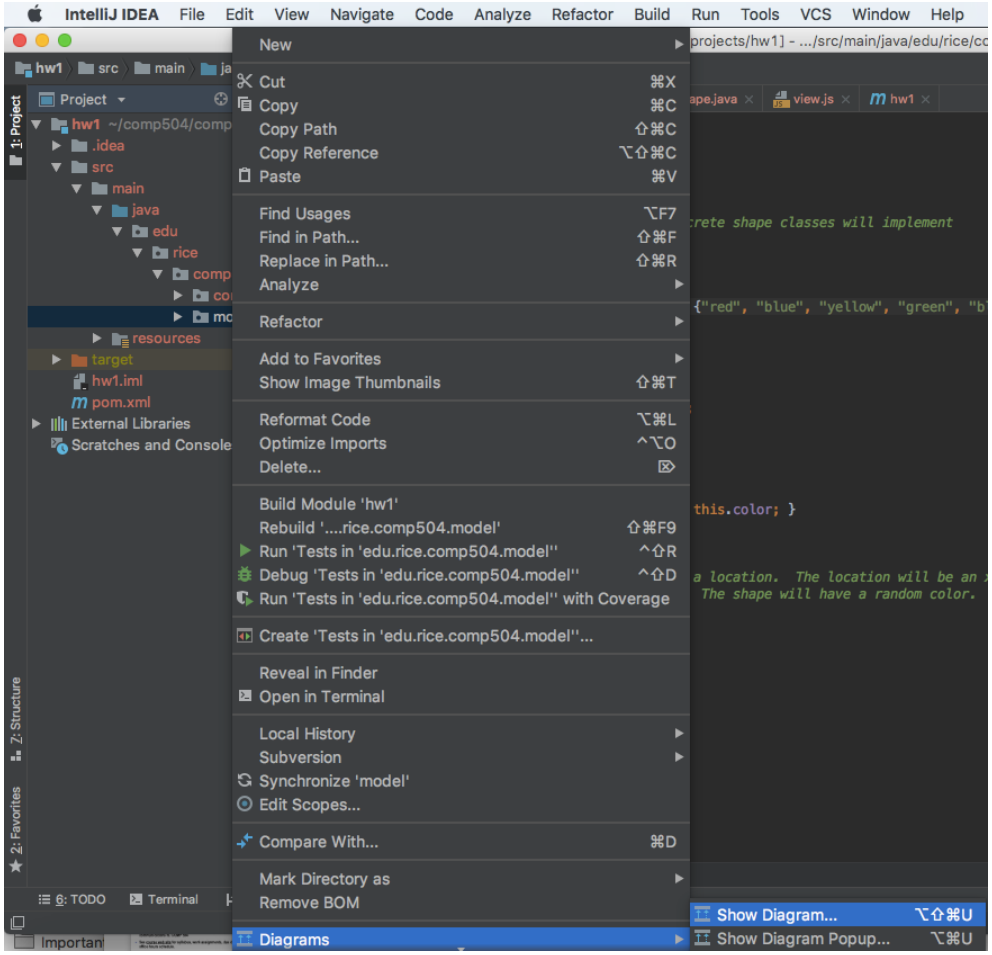
- A use case is an example of how a user might interact with an application
- Design and develop an application by collecting as many use cases as possible
- Use case set informs developer what interfaces are needed to build system

UML Diagrams

- Reflect possible use cases in the system
- Use cases drive interface
  - Interface does not constrain use cases
  - Construct use cases before designing system
- Explain how your system works
  - Customer uses UMLs to understand interfaces within system

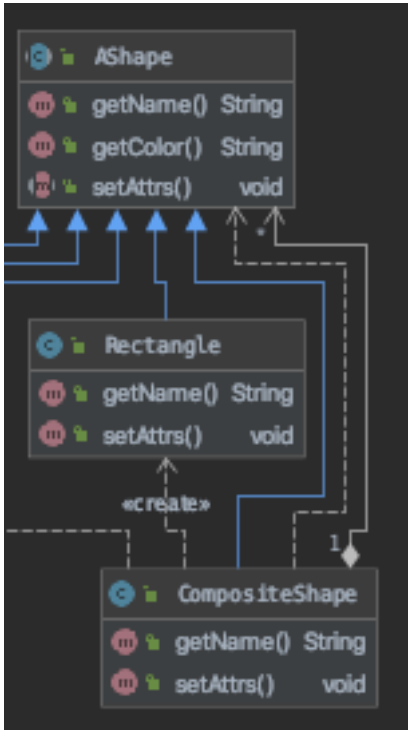
UML Java Diagrams

Unified Modeling Language provides Java class relationship diagram



UML Java Diagrams

- Shows class relationships
- Displays abstract classes/methods
- CompositeShape can have many AShapes



ABall

- Interfaces (abstract classes) are a reflection of key design decisions
- Set after design review
  - cannot be changed (e.g. JDK)
  - companies design software based on initial interface released
- Use cases are critical to project software design/development process

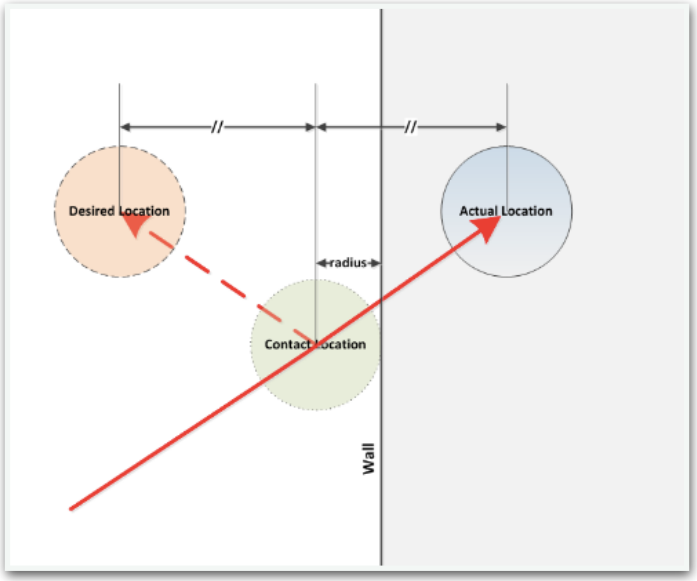
```
/**
 * Ball listens for property changes.  Ball updates itself in the Ball World based on it's type.
 */
public abstract class ABall implements PropertyChangeListener {
    private Point loc;
    private String color;
    private int radius;
    private Point vel;
    private String type;

    /**
     * Constructor for ABall
     * @param loc The ball location.  The origin (0,0) is the upper left corner of the canvas.
     * @param radius  The ball radius.
     * @param vel The ball velocity.  The velocity is a vector with an x and y component.
     * @param color The ball color.
     * @param type The ball type.
     */
    public ABall(Point loc, int radius, Point vel, String color, String type) {
        this.loc = loc;
        this.radius = radius;
        this.vel = vel;
        this.color = color;
        this.type = type;
    }
```


Ball World

- Multiple moving balls may collide with canvas boundary
  - treat canvas boundary as a wall
- Ball World update should update ball listeners (via adapter)
  - view determines when periodic update occurs
- Utilize PropertyChangeSupport/Listener framework to communicate with a ball group in Ball World
  - PCS notifies PCLs when property change has occurred
- The DispatchAdapter has the PCS object

Collisions

- Assume ball does not slow down while moving or colliding with a wall
- Negate x velocity when hitting right or left wall
- Negate y velocity when hitting top or bottom wall



Rotations

- Some assignments may require at least one rotating ball

$$V_{x,\,new} = V_{x,\,old}\ cos(\theta) - V_{y,\,old}\ sin(\theta)$$

$$V_{y,\,new} = V_{y,\,old}\ cos(\theta) + V_{x,\,old}\ sin(\theta)$$

- Change ball rotation without changing velocity

Worksheet #5: Design for Scaling

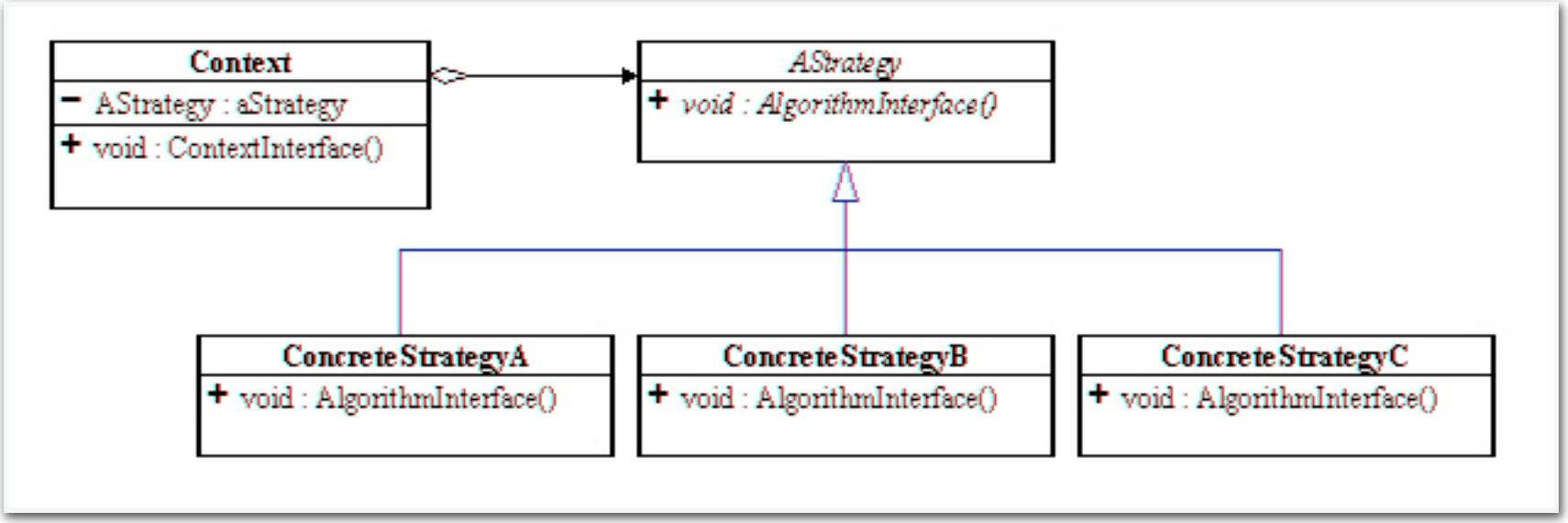
Assume your HW #1 needs to now support 12 shapes in addition to a composite shape. What would you need to do to scale your solution? Could it be done quickly?

Assume there's no restriction on what the view can have (buttons, select, etc…).

What would you need to change in the model, view, controller?

Strategy Design Pattern

- A *strategy* implements the behavior of the *Context* class
  - the context is a concrete class
  - strategy should be connected to a context
- The context is invariant, the strategy (behavior) is variant
- The strategy design pattern uses the union design pattern



Strategy Design Pattern Principles

- Decouple algorithm from host. Encapsulate algorithm in separate class.
- Good to use when the objects are basically the same. Differ in behavior
- Reduce several objects to one object that uses several strategies

IUpdateStrategy

- All concrete strategies can implement *updateState* method

```
public interface IUpdateStrategy {
    public void updateState(Ball context);
}
```

- Apply strategy to context (ball)
  - separate behavior from object
  - designed for flexibility
  - allows for context to have multiple strategies

Switching Strategies

- Strategies represent object behavior
  - Flexibility allows for objects to change behaviors
- Updating an object (ball) could cause the behavior to change
  - Switch strategies during some point of time in the Ball World
- How do you switch the strategy of a ball or multiple balls in the Ball World?

POST Request

- Client may not want to send information in the url as a query parameter
  - Data may be sensitive
  - Assume strategies are data not to be shared

- Use POST to send data from the client (view) to the server (controller)

- Data will appear in the payload (request body)
  - Can't just type the url into the browser (not found error)
  - When parsing data, type is String

<select>

- Dropdown box instead of a button

- Allows user the ability select option(s)
  - By default, user can select one option
  - Has attribute *multiple* to select multiple options.

- Use <*option*> tag inside <*select*> to specify choices

Announcements & Reminders

- HW #2 is available and is due Friday Sept 18th at 11:59pm

  Github hw 2: *https://classroom.github.com/a/V3xAEMEY*

- Use Piazza (public or private posts, as appropriate) for all communications re. COMP 504
- See course web site for syllabus, work assignments, due dates, office hours schedule.