Lecture 22

ECE 1145: Software Construction and Evolution

HotCiv GUI (Iteration 8)
(CH 36.7)

Announcements

- Iteration 8 (last one!): Frameworks and MiniDraw due Dec. 12
  - My recommendations:
    - **THIS WEEK**: Frameworks (36.36), MiniDraw Integration (test out gradle tasks), Subject behavior (36.37), Observer updates (36.38)
    - **NEXT WEEK**: Tool development (36.39-40, 42-44), SemiCiv GUI
- OMET Teaching Surveys open until Dec. 12
  - Link in Canvas navigation
- Office Hours Thurs. Dec. 2 10:30 – 11:00 AM
- Final Exam: 12:00 AM Wed. Dec. 15 – 11:59 PM Fri Dec. 17 (similar format as midterm)

Questions for Today

How do we use and combine frameworks with compositional design?

HotCiv + MiniDraw

HotCiv + MiniDraw

"TDD" and Stubs

- Develop the GUI using TDD
- Keep focus & take small steps
- Can't automate all tests
- Use **test stubs**

Integration and System testing

- SemiCiv GUI

HotCiv + MiniDraw

"TDD" and Stubs

- Develop the GUI using TDD
- Keep focus & take small steps
- Can't automate all tests
- Use **test stubs**

Integration and System testing

- SemiCiv GUI

Avoid **"Big Bang"** Integration/Testing!

- Definition: Doing and then testing everything, all at once
- Result: Nothing works, no one knows why

HotCiv + MiniDraw

MiniDraw Merging

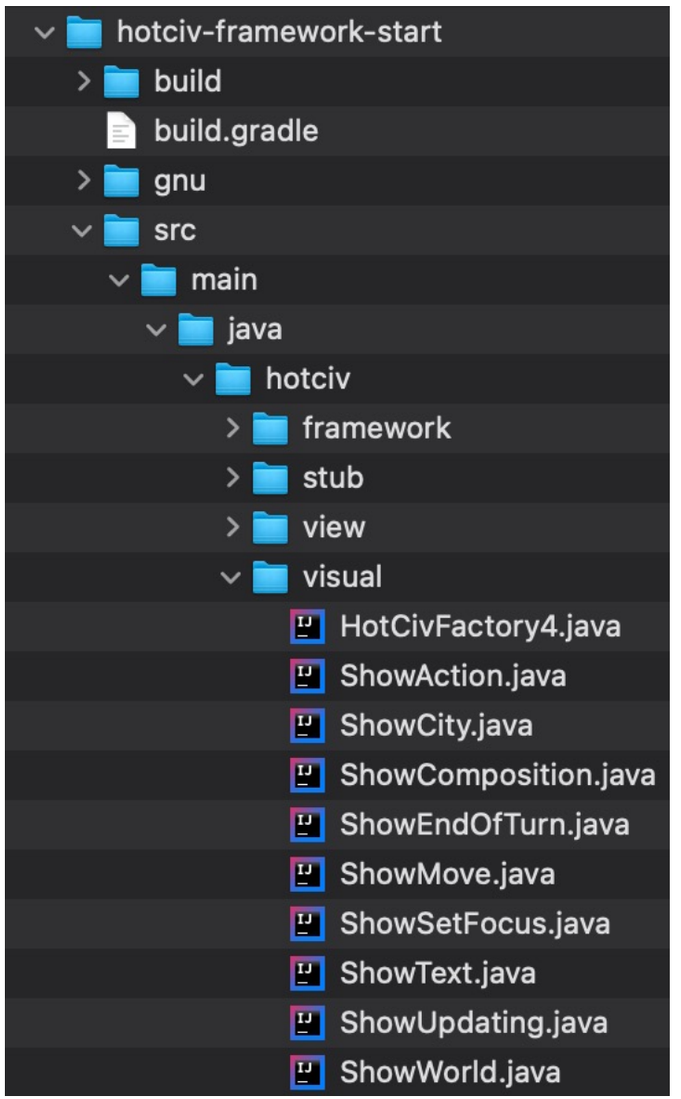
- Updated gradle build script (build.gradle) with more targets!

```
44  task show(type: JavaExec) {
45      group 'HotCiv Demonstration'
46      description 'Demonstrate MapView'
47
48      main = 'hotciv.visual.ShowWorld'
49      classpath = sourceSets.main.runtimeClasspath
50  }
51
52  task text(type: JavaExec) {
53      group 'HotCiv Demonstration'
54      description 'Demonstrate TextFigure'
55
56      main = 'hotciv.visual.ShowText'
57      classpath = sourceSets.main.runtimeClasspath
58  }
59
60  task city(type: JavaExec) {
61      group 'HotCiv Demonstration'
62      description 'Demonstrate CityFigure'
63
64      main = 'hotciv.visual.ShowCity'
65      classpath = sourceSets.main.runtimeClasspath
66  }
```

HotCiv + MiniDraw

MiniDraw Merging

- Updated gradle build script (build.gradle) with more targets!
- Additional code (visual demos)



HotCiv + MiniDraw

MiniDraw Merging

- Updated gradle build script (build.gradle) with more targets!
- Additional code (visual demos)
- Minor changes in interface:
  - Game has two ‘observer’ methods
  - GameObserver.java

Game.java

```java
/** add an observer on this game instance. The game
 * instance acts as 'subject' in the pattern.
 * @param observer the observer to notify in case of
 * state changes.
 */
public void addObserver(GameObserver observer);

/** set the focus on a specific tile. This will
 * result in an event being broadcast to all
 * observers that focus has been changed to
 * this tile. Precondition: the position
 * is within the limits of the game world.
 * @param position the position of the tile that
 * has focus.
 */
public void setTileFocus(Position position);
```

GameObserver.java

```java
public interface GameObserver {
 /** invoked every time some change occurs on a position
  * in the world - a unit disappears or appears, a
  * city appears, a city changes player color, or any
  * other event that requires the GUI to redraw the
  * graphics on a particular position.
  * @param pos the position in the world that has changed state
  */
 public void worldChangedAt(Position pos);

 /** invoked just after the game's end of turn is called
  * to signal the new "player in turn" and world age state.
  * @param nextPlayer the next player that may move units etc.
  * @param age the present age of the world
  */
 public void turnEnds(Player nextPlayer, int age);

 /** invoked whenever the user changes focus to another
  * tile (for inspecting the tile's unit and city
  * properties.)
  * @param position the position of the tile that is
  * now inspected/has focus.
  */
 public void tileFocusChangedAt(Position position);
}
```

HotCiv + MiniDraw

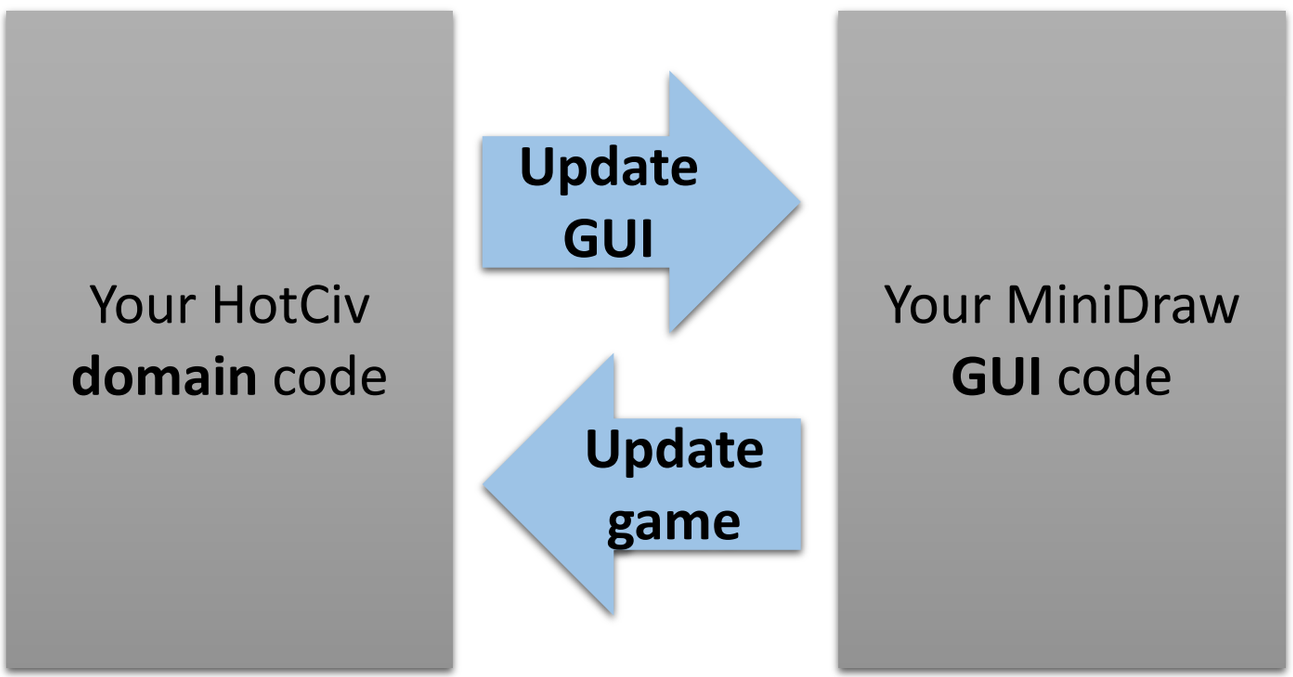
Merge the provided code **carefully**, in a **separate branch**, following instructions in Iteration 8.

HotCiv + MiniDraw

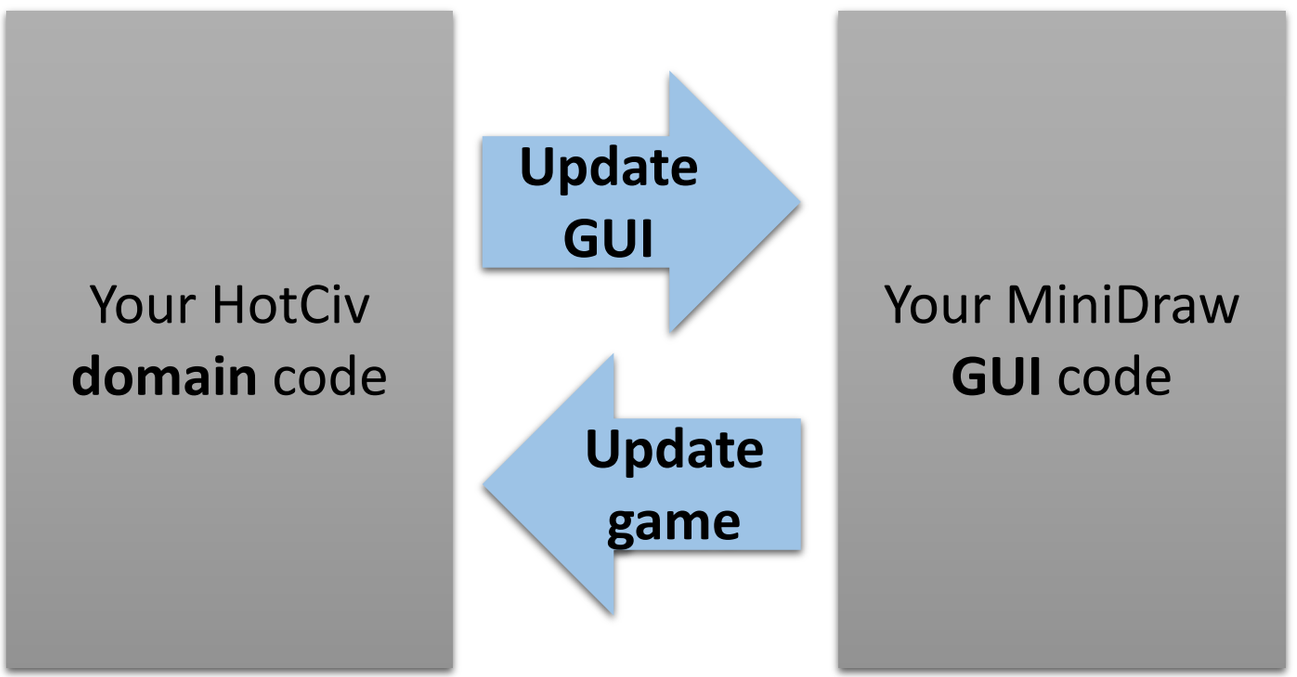
Merge the provided code **carefully**, in a **separate branch**, following instructions in Iteration 8.

1. Create a new branch of your HotCiv code base
2. Unzip starter code **in a separate directory**, test out new gradle tasks
3. Copy new parts of build.gradle into your HotCiv branch
   - Run ‘gradle clean test’ to make sure it works
4. Copy “resources” (graphics) into your HotCiv, maintaining directory structure
5. Copy GameObserver.java – should still compile
6. Copy “stub”, “view”, and “visual” folders into your HotCiv, maintaining directory structure – will cause compilation errors
7. Copy two new Game.java methods – update your GameImpl with empty method implementations

HotCiv + MiniDraw



HotCiv + MiniDraw



Observer pattern
[Domain *informs* GUI]

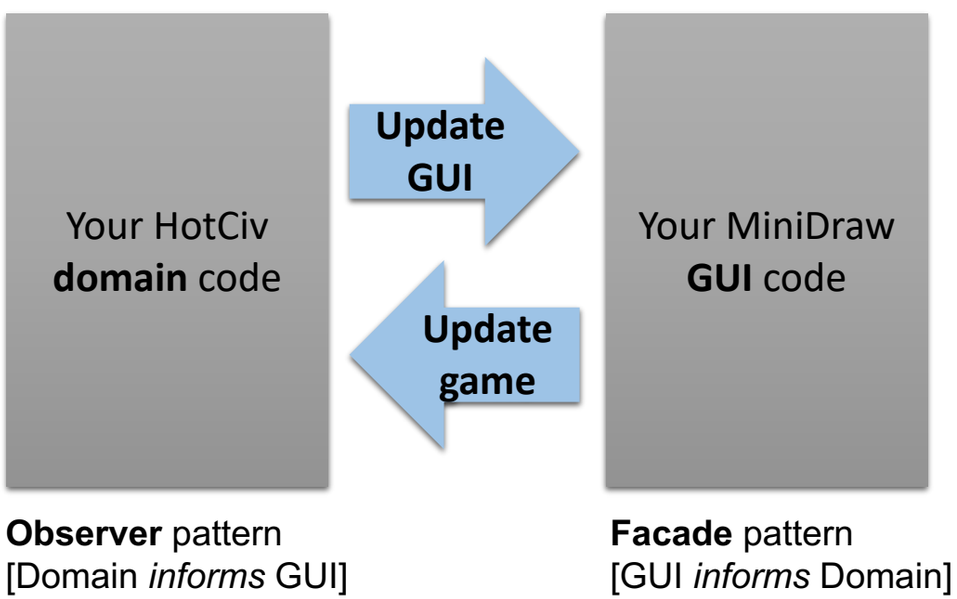
Facade pattern
[GUI *informs* Domain]

HotCiv + MiniDraw

Only rely on **interfaces**

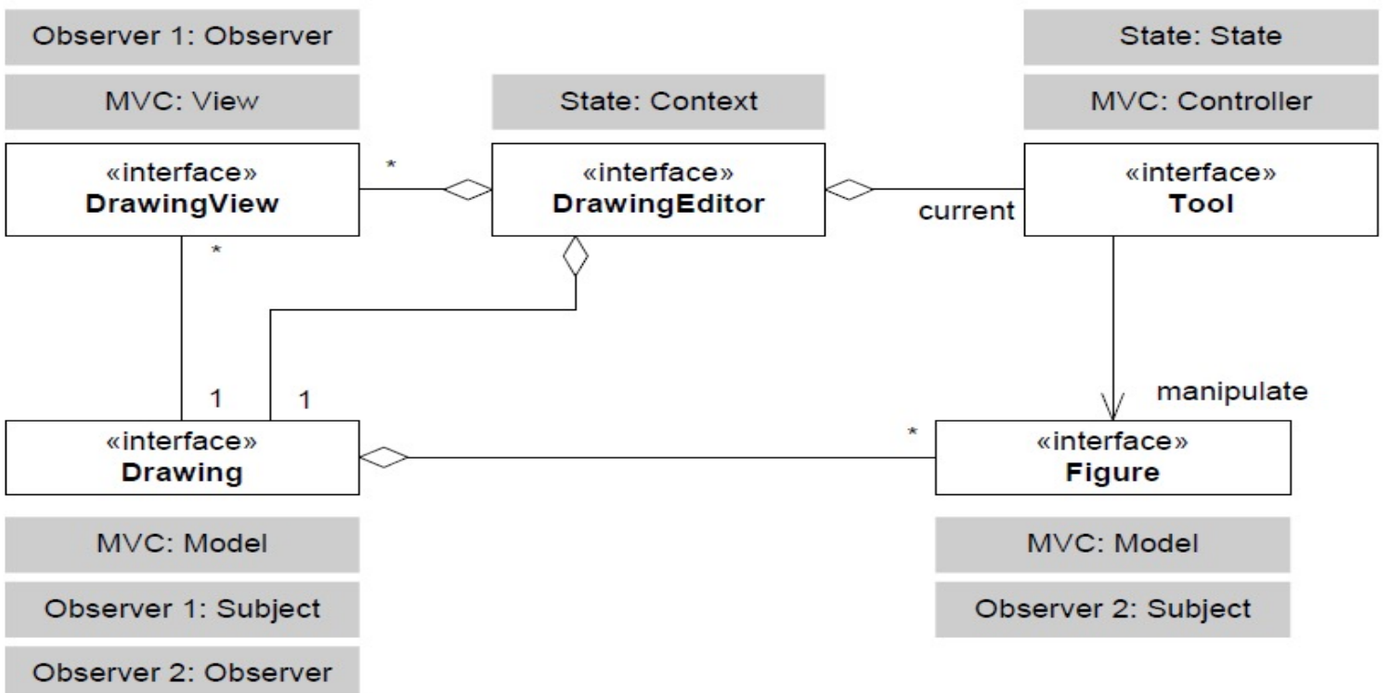
- Develop the GUI using **Stub implementations** of the domain's Facade and Observer roles
  - Facade = Interface Game
  - Observer = Interface GameObserver

Reuse MiniDraw's graphical abilities



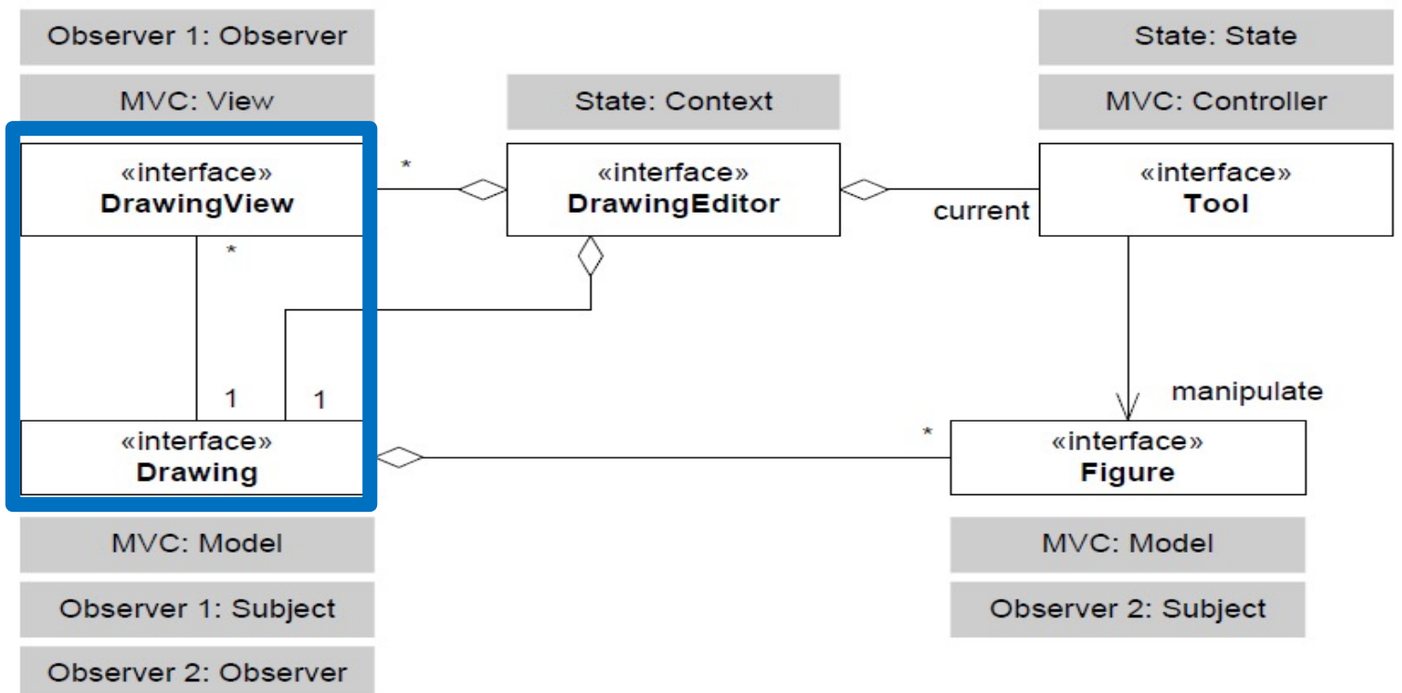
HotCiv + MiniDraw

MiniDraw Review



HotCiv + MiniDraw

MiniDraw Review



Model-View (Observer)

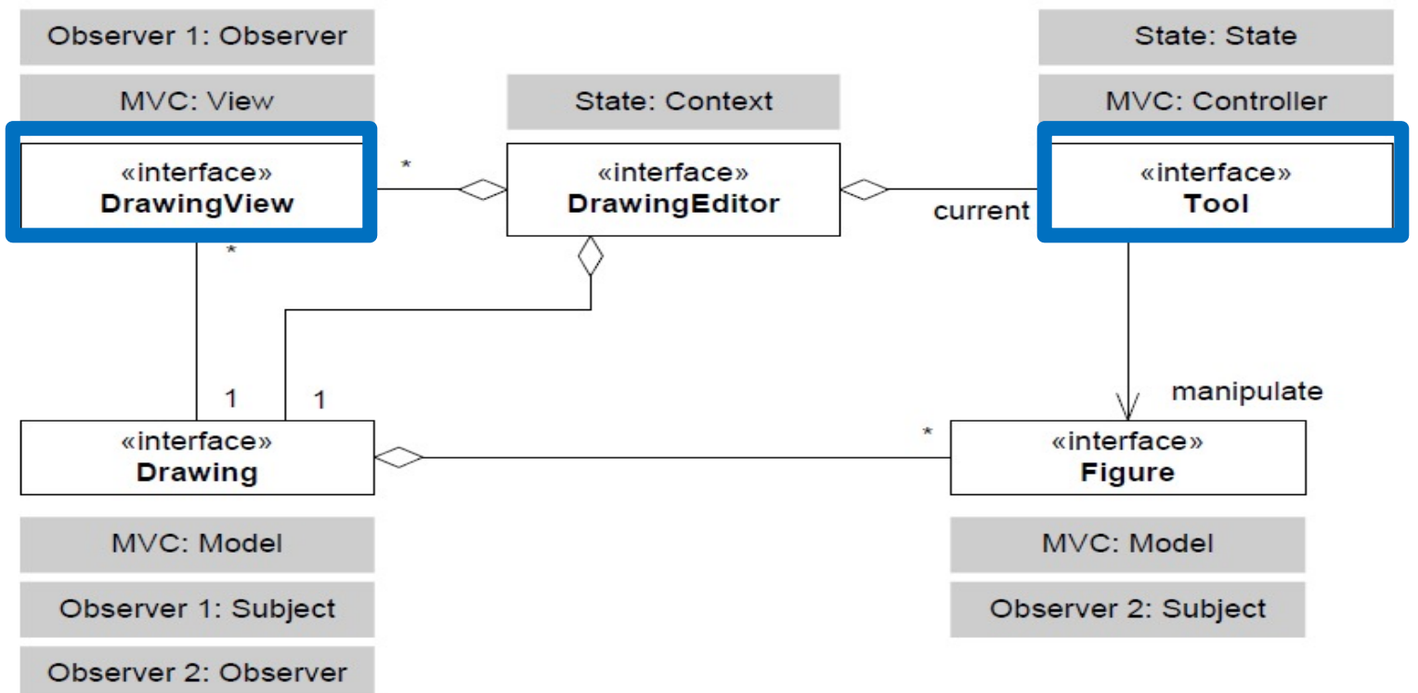
③ Data **visualization** is the behavior that varies

① Responsibility to visualize data is expressed in an interface: **View**

② Instead of the data object (model) itself being responsible for drawing graphics, it lets someone else do the job: the **views**

HotCiv + MiniDraw

MiniDraw Review



View-Controller (State)

③ Data **manipulation** is the behavior that varies

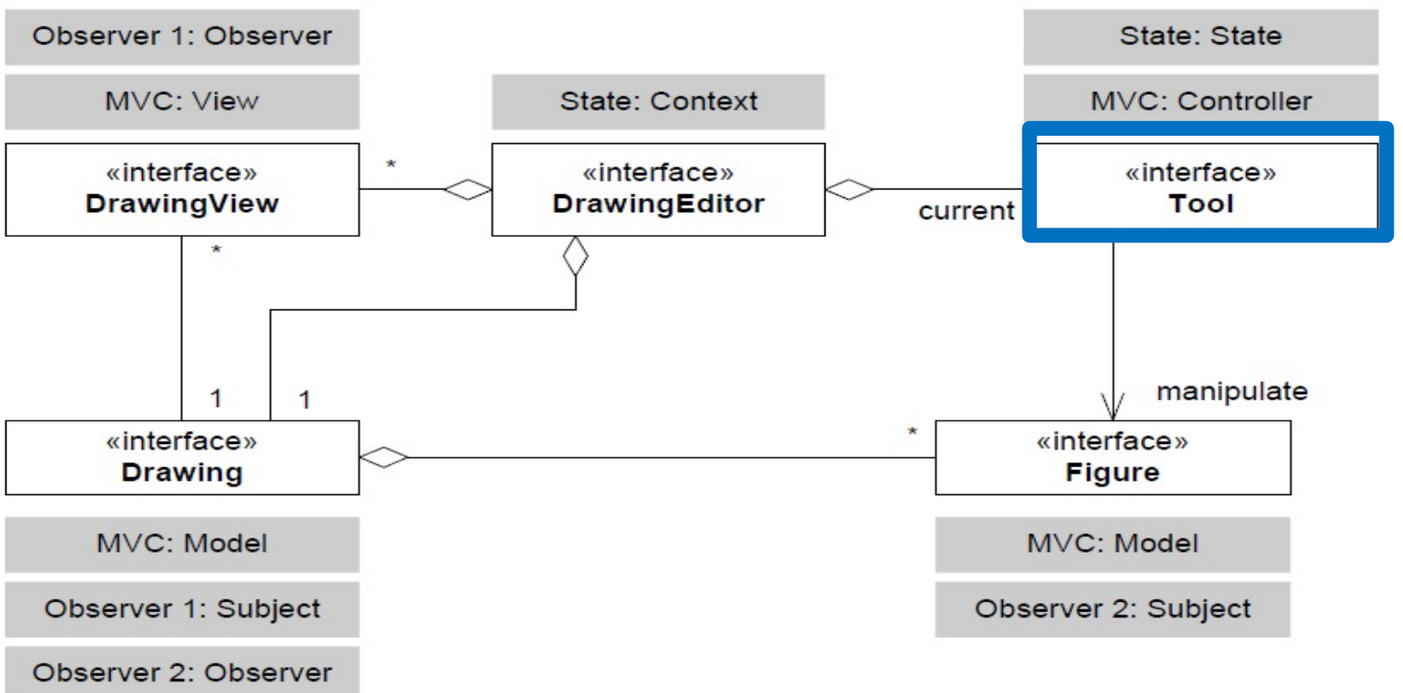
① Responsibility to manipulate data is expressed in an interface: **Controller**

② Instead of the graphical view itself being responsible for manipulating data, it lets someone else do the job: the **controller**

HotCiv + MiniDraw

MiniDraw Review

Selected tool defines the state



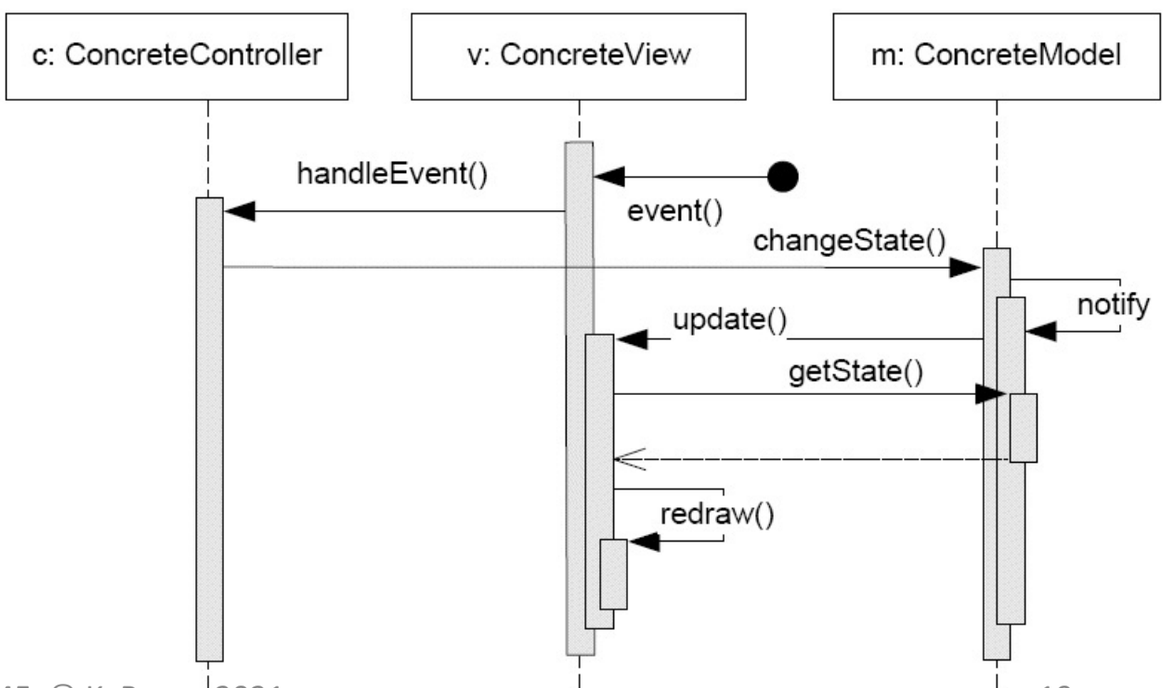
In MiniDraw the **editor** is in a **state** that determines how mouse events are interpreted. Mouse events are forwarded to the editor's **tool**.

→ Change the tool, change how mouse events are interpreted

Review: MVC

To “do things” with the mouse, we need:

- A tool/controller to intercept mouse events
- Controller initiates change in model state
- Model notifies registered observers (views)
  - Observers receive the event…
  - Get any needed state information
  - Redraw (update) the display



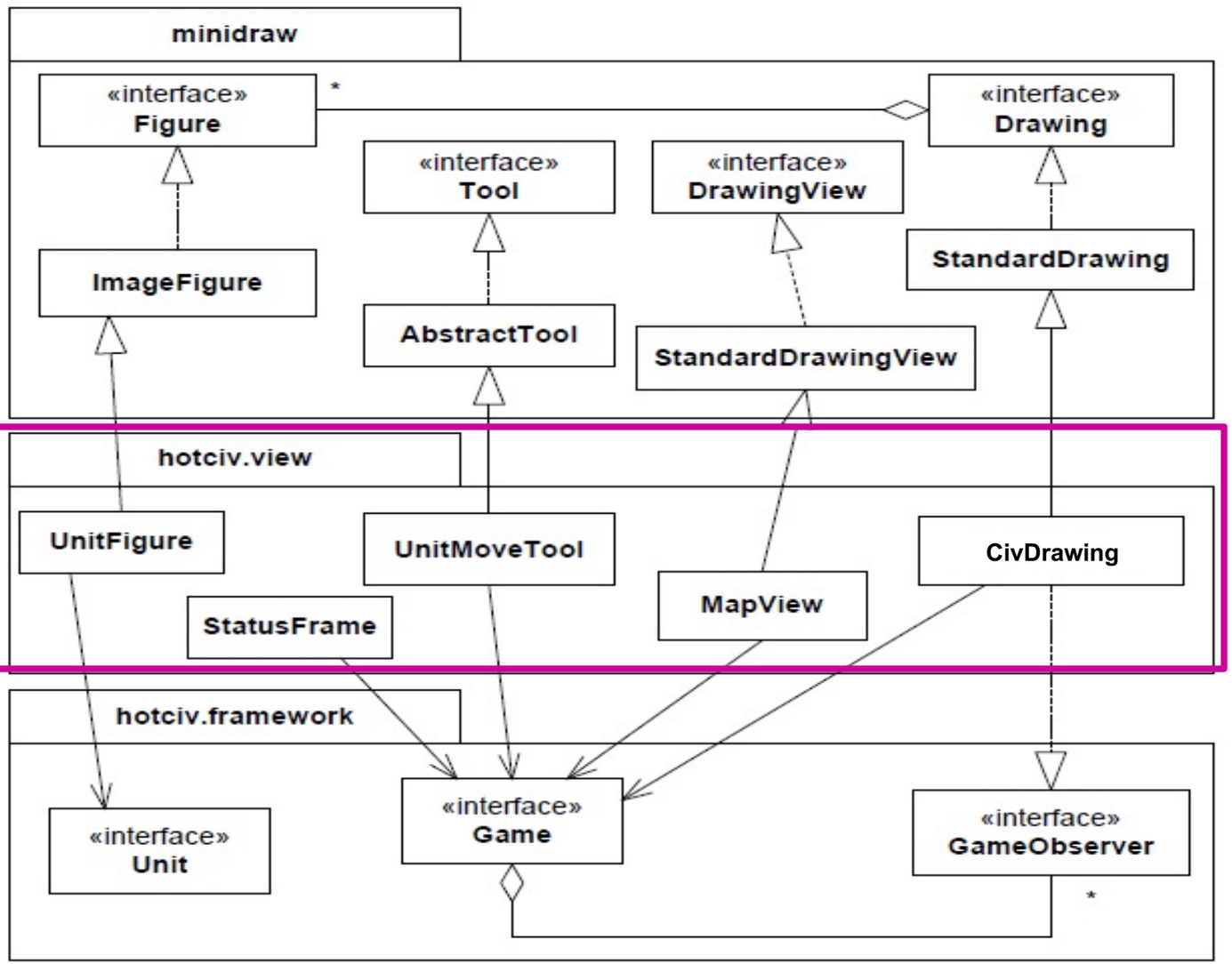
HotCiv + MiniDraw

Three layers:

MiniDraw

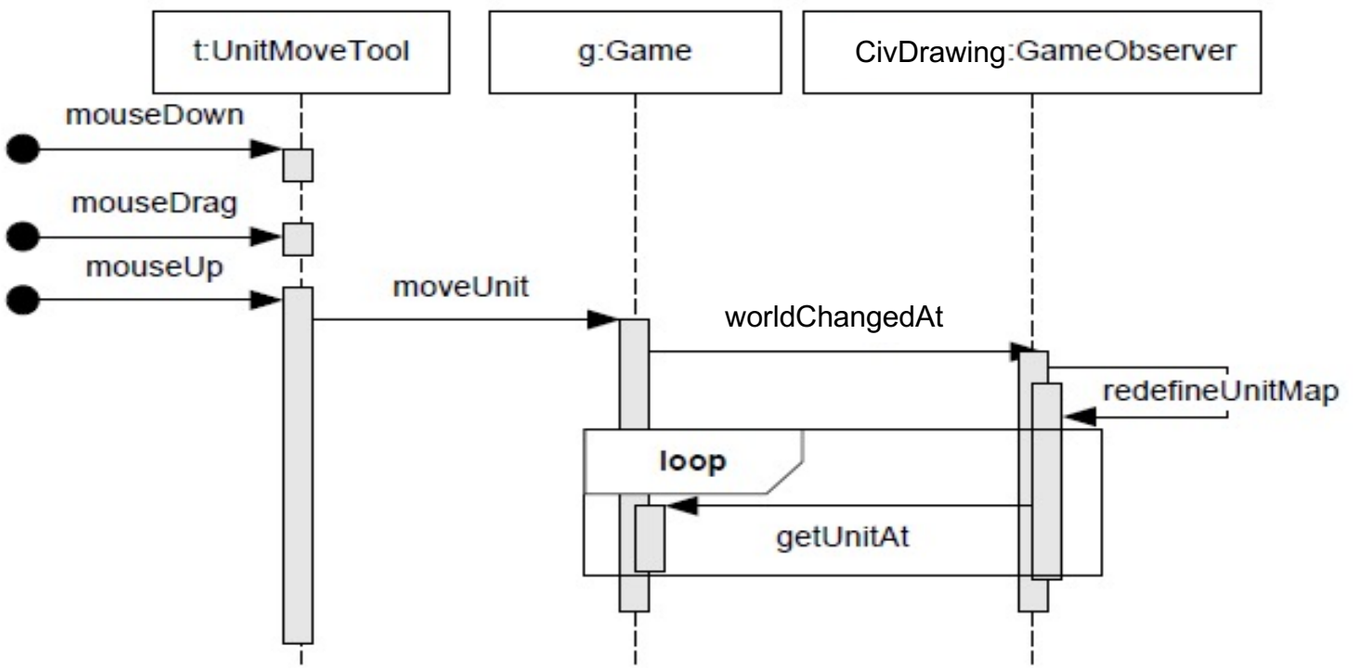
hotciv.view: Hot spots for figures, tools, drawing, views etc

Your HotCiv



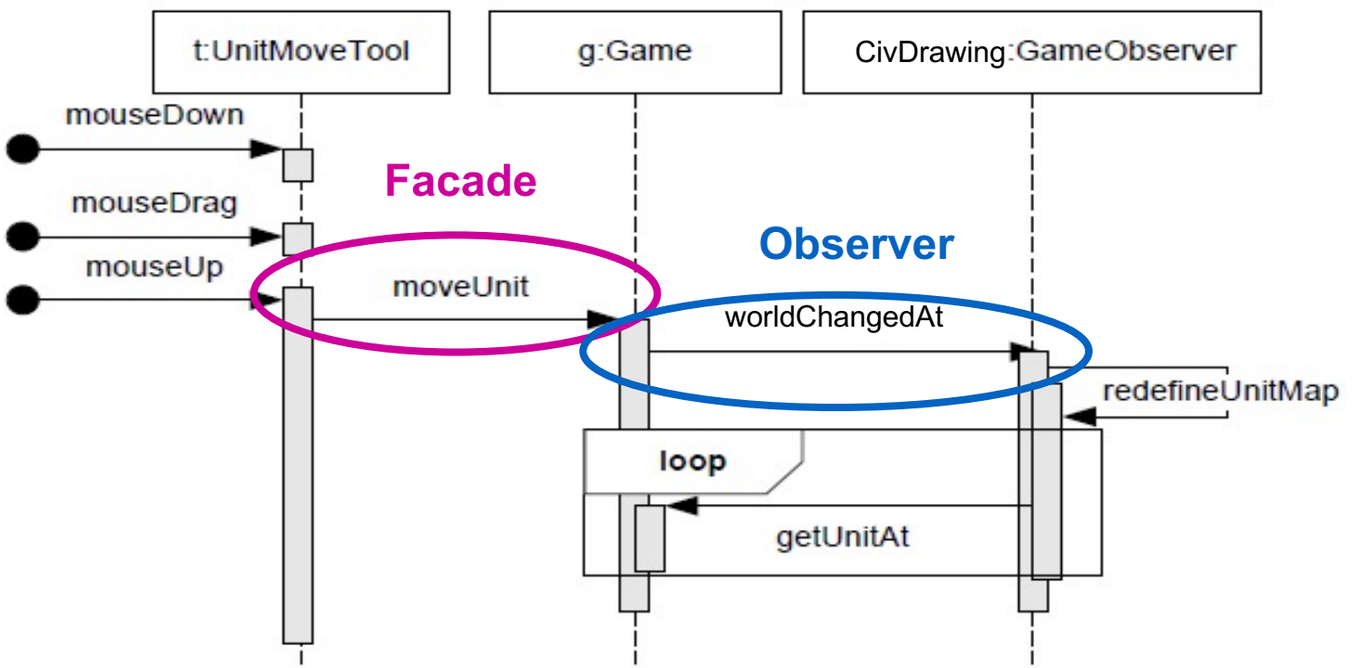
Example: Moving Units

Moving units means invoking Game's "moveUnit"



Provided code partially implements this (see ShowMove.java)

Example: Moving Units

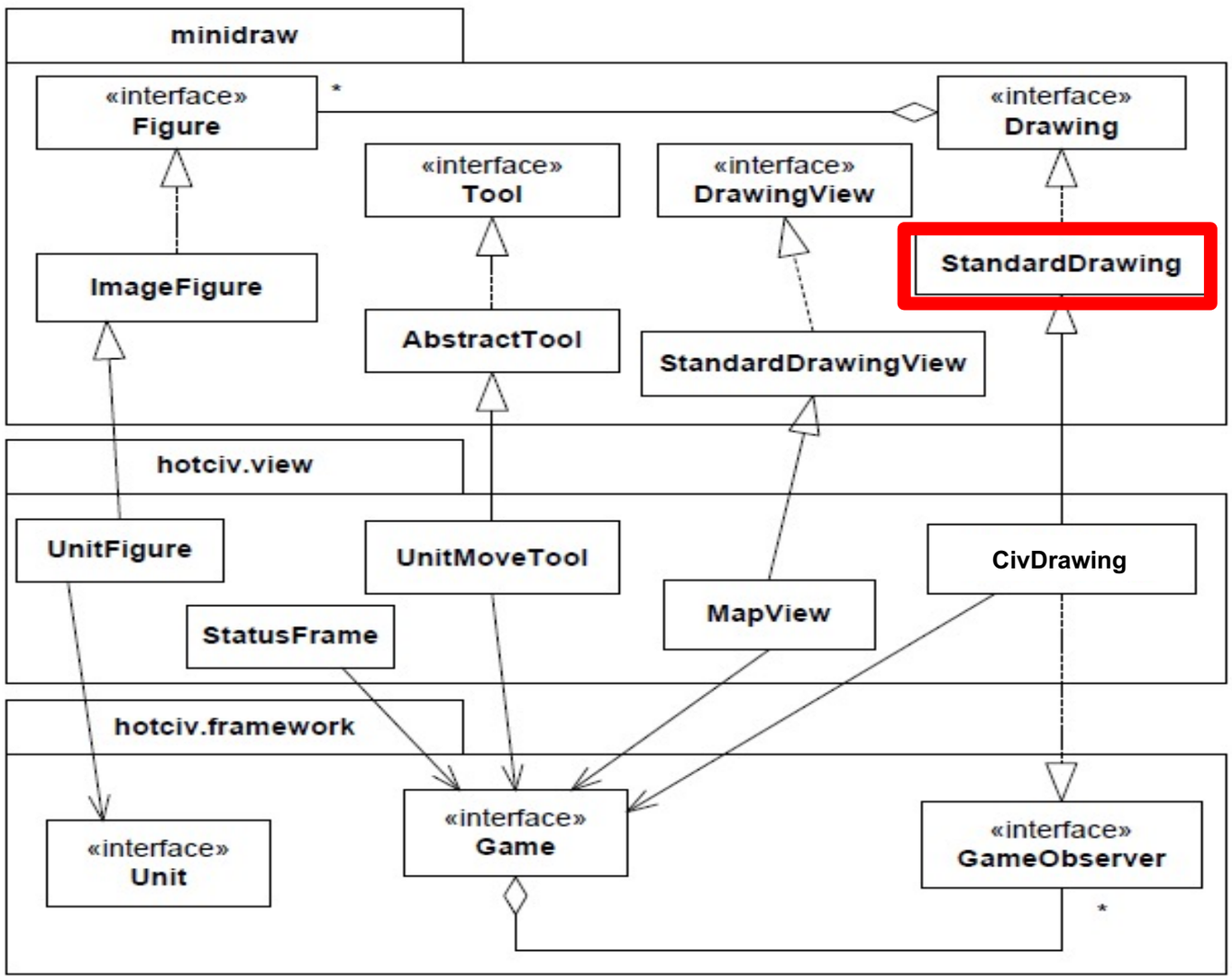


GUI ⇒ Domain ⇒ GUI

HotCiv + MiniDraw

hotciv.view.CivDrawing

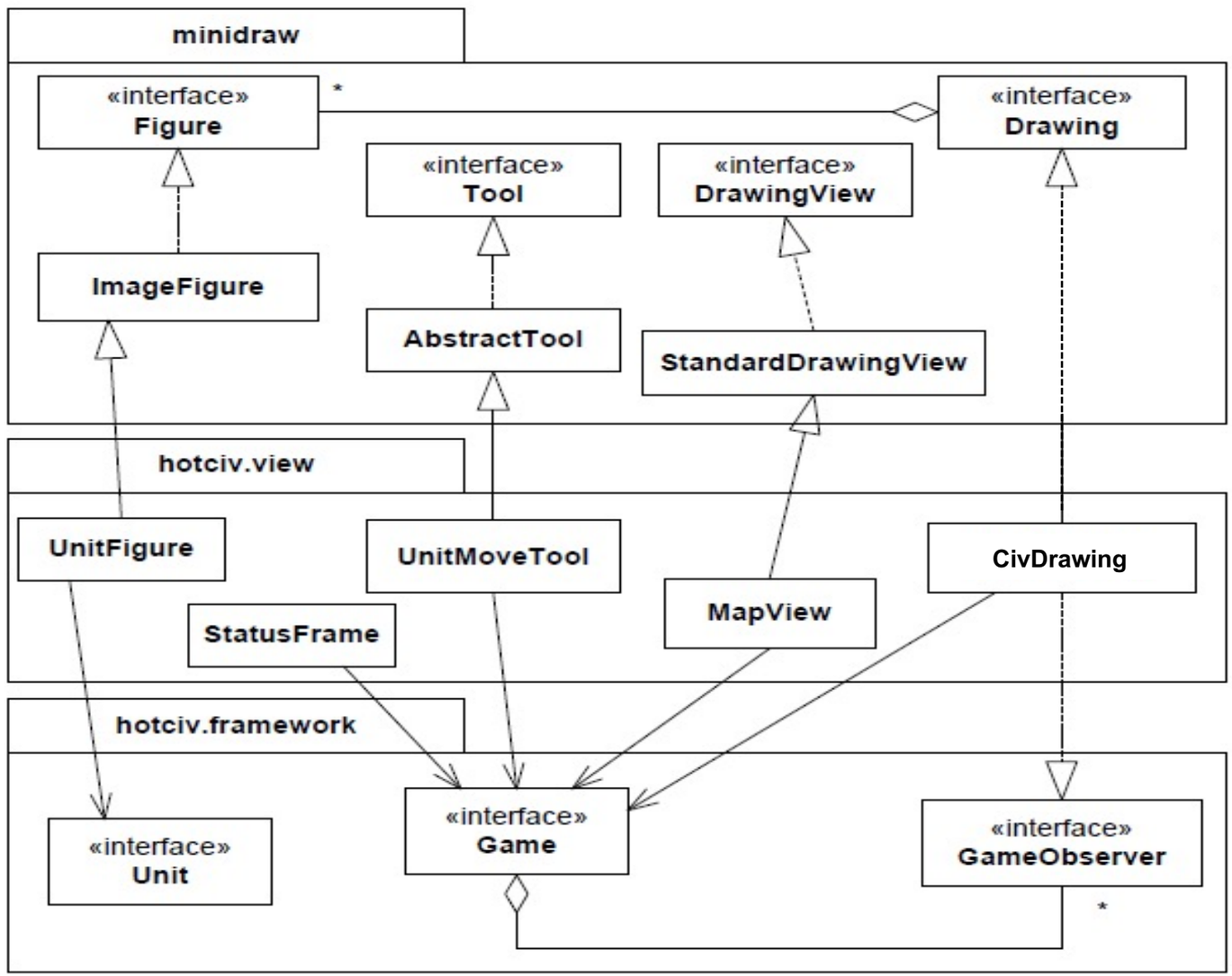
```
public class CivDrawing
  implements Drawing, GameObserver {

  protected Drawing delegate;
  /** store all moveable figures visible in this drawing = units */
  protected Map<Unit,UnitFigure> unitFigureMap;

  /** the Game instance that this CivDrawing is going to render units
   * from */
  protected Game game;

  public CivDrawing( DrawingEditor editor, Game game ) {
    super();
    this.delegate = new StandardDrawing();
    this.game = game;
    this.unitFigureMap = new HashMap<>();

    // register this unit drawing as listener to any game state
    // changes...
    game.addObserver(this);
    // ... and build up the set of figures associated with
    // units in the game.
    defineUnitMap();
    // and the set of 'icons' in the status panel
    defineIcons();
  }
```



HotCiv + MiniDraw

hotciv.view.CivDrawing

```
public class CivDrawing
  implements Drawing, GameObserver {

  protected Drawing delegate;
  /** store all moveable figures visible in this drawing = units */
  protected Map<Unit,UnitFigure> unitFigureMap;

  /** the Game instance that this CivDrawing is going to render units
   * from */
  protected Game game;

  public CivDrawing( DrawingEditor editor, Game game ) {
    super();
    this.delegate = new StandardDrawing();
    this.game = game;
    this.unitFigureMap = new HashMap<>();

    // register this unit drawing as listener to any game state
    // changes...
    game.addObserver(this);
    // ... and build up the set of figures associated with
    // units in the game.
    defineUnitMap();
    // and the set of 'icons' in the status panel
    defineIcons();
  }
```

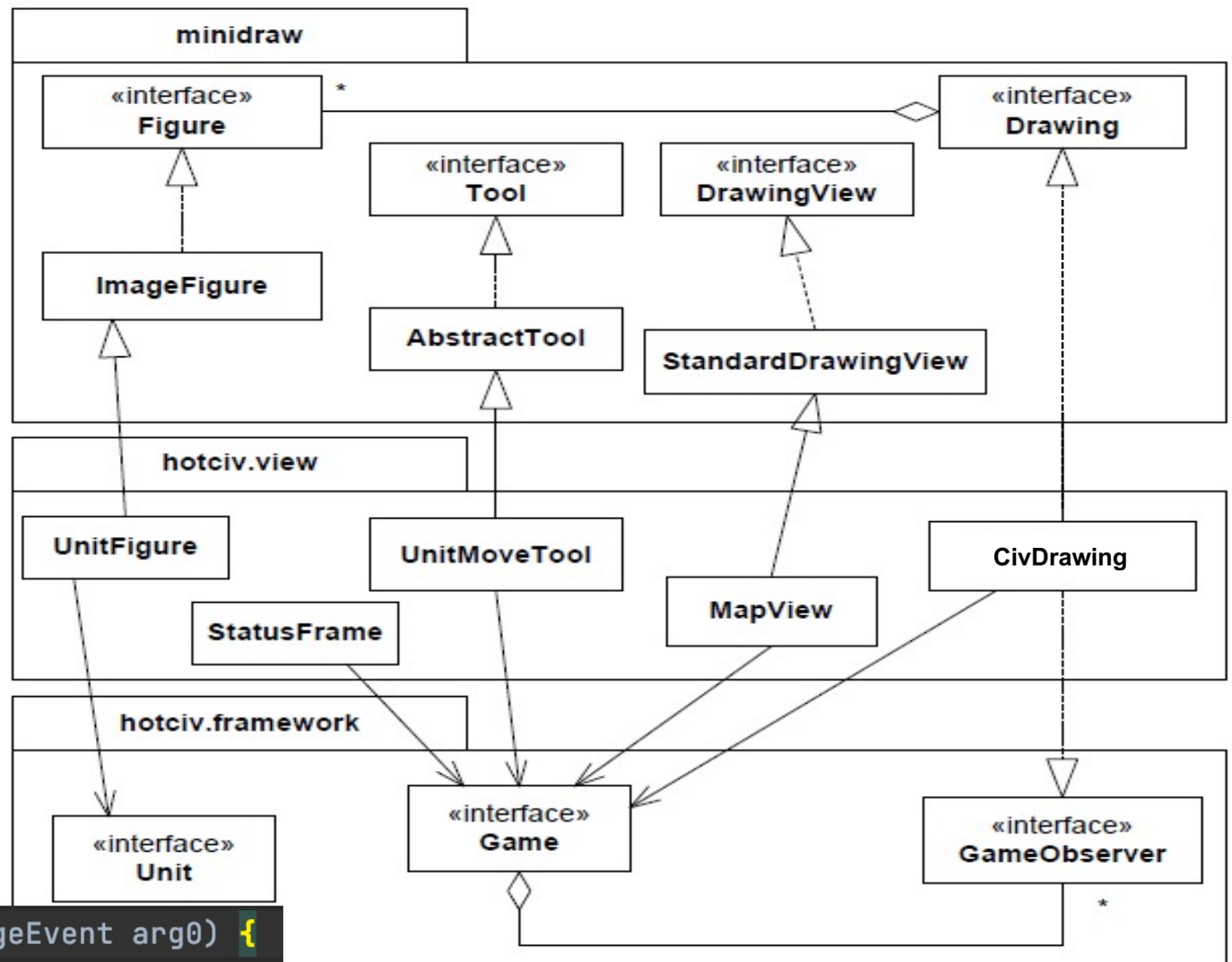


HotCiv + MiniDraw

hotciv.view.CivDrawing

```
public class CivDrawing
  implements Drawing, GameObserver {

  protected Drawing delegate;
  /** store all moveable figures visible in this drawing = units */
  protected Map<Unit,UnitFigure> unitFigureMap;

  /** the Game instance that this CivDrawing is going to render units
   * from */
  protected Game game;

  public CivDrawing( DrawingEditor editor, Game game ) {
    super();
    this.delegate = new StandardDrawing();
    this.game = game;
    this.unitFigureMap = new HashMap<>();

    // register this unit drawing as listener to any game state
    // changes...
    game.addObserver(this);
    // ... and build up the set of figures associated with
    // units in the game.
    defineUnitMap();
    // and the set of 'icons' in the status panel
    defineIcons();
  }
```

```
public void figureChanged(FigureChangeEvent arg0) {
  delegate.figureChanged(arg0);
}
```



Iteration 8: Four Part Integration

- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).
- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.

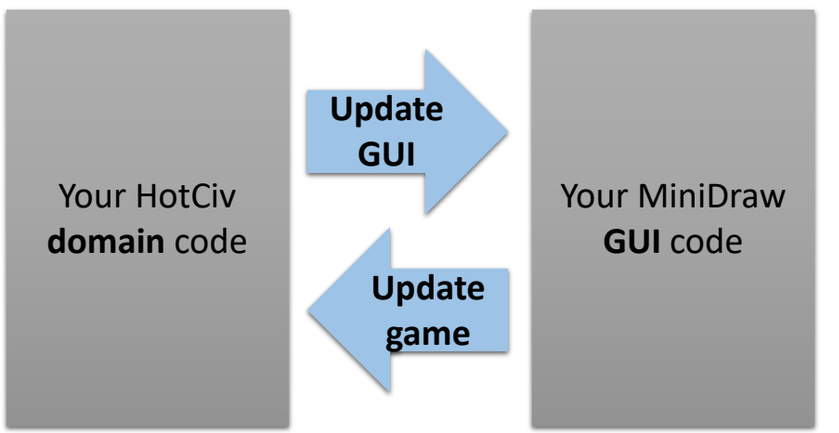
Iteration 8: Integration Part 1

- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).
- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.

Iteration 8: Observer (Subject)

Focus:

- Your HotCiv game must serve as **Subject** for the GUI to get events when state changes (units move, cities are created, etc.)
- **Domain informs GUI**



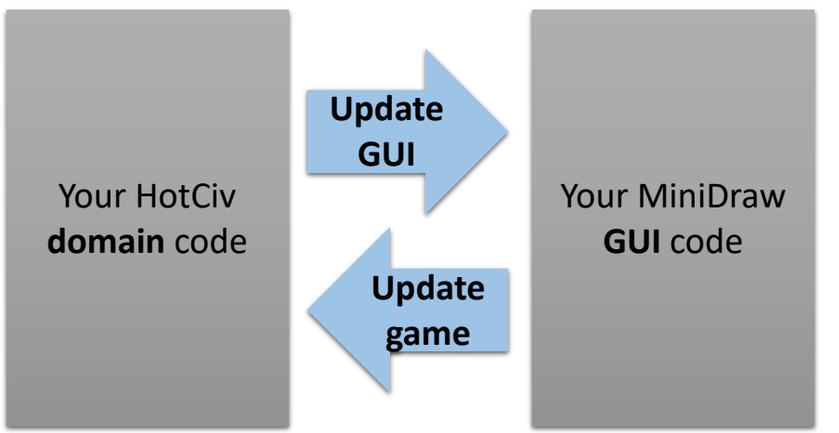
Observer pattern
[Domain *informs* GUI]

Facade pattern
[GUI *informs* Domain]

Iteration 8: Observer (Subject)

Focus:

- Your HotCiv game must serve as **Subject** for the GUI to get events when state changes (units move, cities are created, etc.)
- **Domain informs GUI**

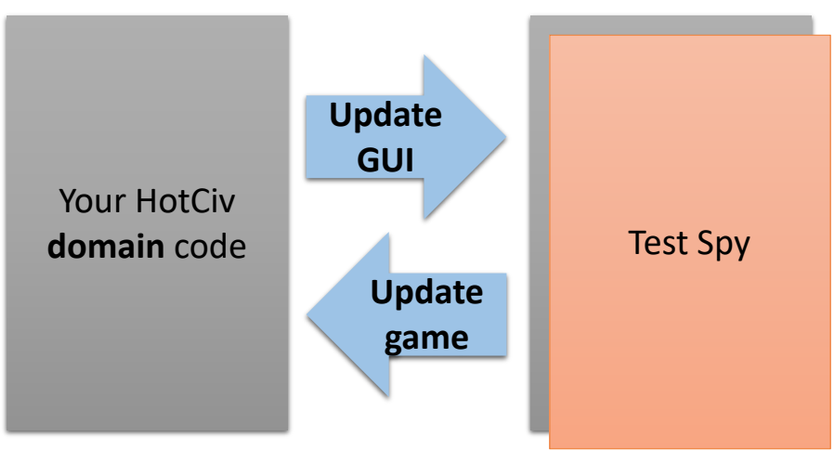


Observer pattern
[Domain *informs* GUI]

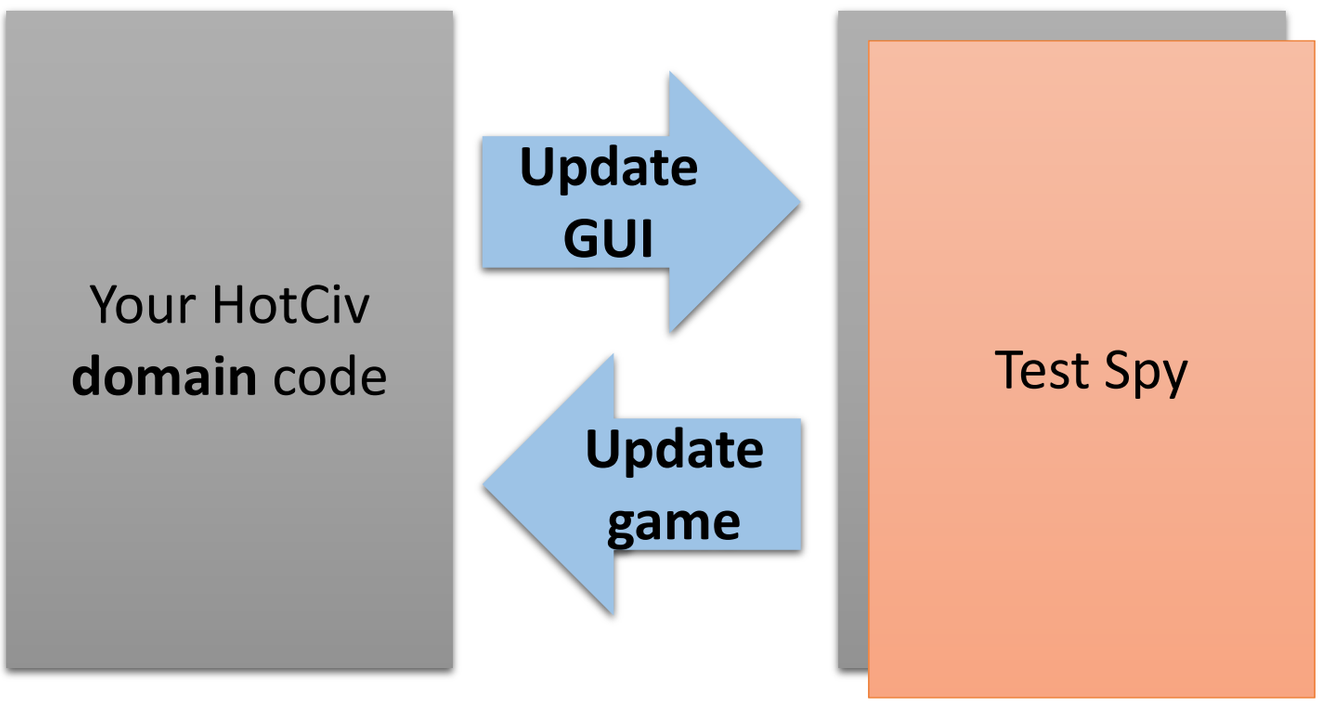
Facade pattern
[GUI *informs* Domain]

Can be done with TDD using **Automated Testing**:

- Make a Test Stub GameObserver implementation
  - Actually a Test Spy which records all calls
- Inject it into your GameImpl
  - Implement Game.addObserver
- Validate that the GameObserver implementation gets called correctly when Game state changes
  - moveUnit, endOfTurn, etc.



Iteration 8: Observer (Subject)



Iteration 8: Observer (Subject)

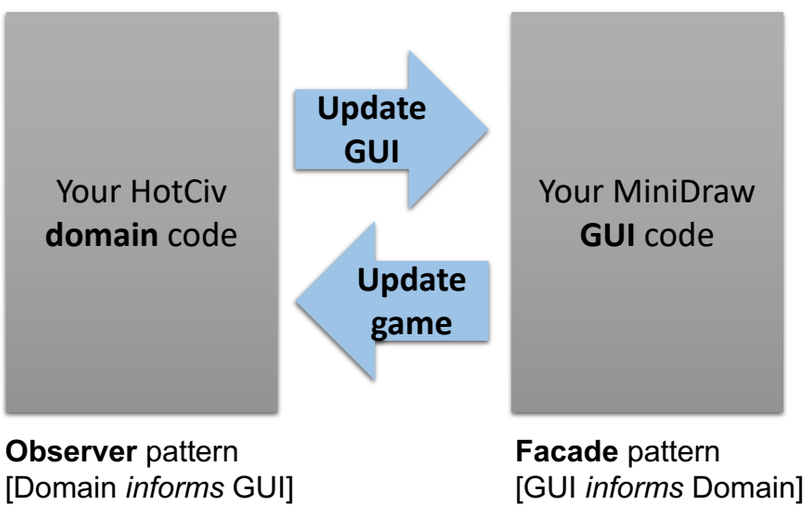
Iteration 8: Integration Part 2

- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).
- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.

Iteration 8: Observer (Update)

Focus:

- All Domain events must be reflected correctly in the MiniDraw GUI so the graphics are synchronized with the game's state
- **Domain informs GUI**
  - Archer moved, update graphics, etc.



Iteration 8: Observer (Update)

Focus:

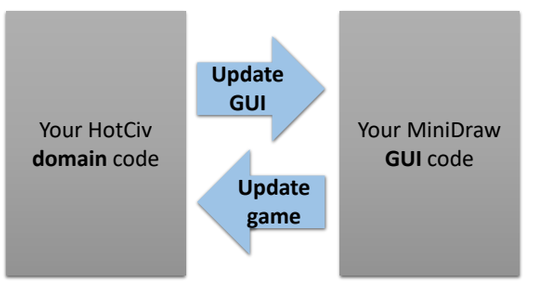
- All Domain events must be reflected correctly in the MiniDraw GUI so the graphics are synchronized with the game's state
- **Domain informs GUI**
  - Archer moved, update graphics, etc.

Testing **cannot** be automated!

→ Use manual testing (visual inspection)

But, avoid **big bang** testing

- Use a Game test stub!

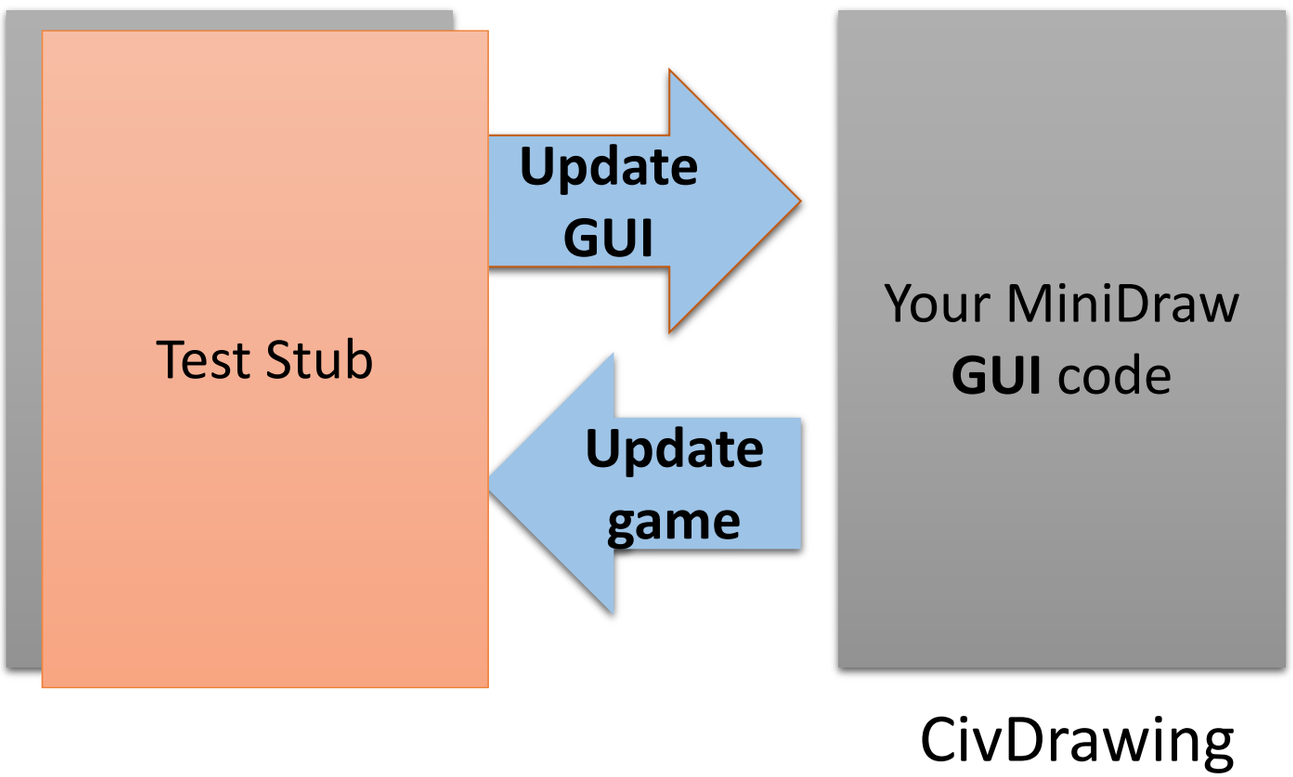


Observer pattern
[Domain *informs* GUI]

Facade pattern
[GUI *informs* Domain]

Iteration 8: Observer (Update)

Use a StubGame, not the real HotCiv GameImpl:



Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

```
public class ShowUpdating {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to see Drawing updates",
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes reflected on the GUI");

    // Try to set the selection tool instead to see
    // completely free movement of figures, including the icon

    // editor.setTool( new SelectionTool(editor) );
  }
}
```

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.stub.StubGame2

```java
public class StubGame2 implements Game {

  // === Unit handling ===
  private Position pos_archer_red;
  private Position pos_legion_blue;
  private Position pos_settler_red;
  private Position pos_ufo_red;

  private Unit red_archer;

  public Unit getUnitAt(Position p) {
    if ( p.equals(pos_archer_red) ) {
      return red_archer;
    }
    if ( p.equals(pos_settler_red) ) {
      return new StubUnit( GameConstants.SETTLER, Player.RED );
    }
    if ( p.equals(pos_legion_blue) ) {
      return new StubUnit( GameConstants.LEGION, Player.BLUE );
    }
    if ( p.equals(pos_ufo_red) ) {
      return new StubUnit( ThetaConstants.UFO, Player.RED );
    }
    return null;
  }
```

```java
  // Stub only allows moving red archer
  public boolean moveUnit( Position from, Position to ) {
    System.out.println( "-- StubGame2 / moveUnit called: "+from+"->"+to );
    if ( from.equals(pos_archer_red) ) {
      pos_archer_red = to;
    }
    // notify our observer(s) about the changes on the tiles
    gameObserver.worldChangedAt(from);
    gameObserver.worldChangedAt(to);
    return true;
  }

  // === Turn handling ===
  private Player inTurn;
  public void endOfTurn() {
    System.out.println( "-- StubGame2 / endOfTurn called." );
    inTurn = (getPlayerInTurn() == Player.RED ?
              Player.BLUE :
              Player.RED );
    // no age increments
    gameObserver.turnEnds(inTurn, age: -4000);
  }
  public Player getPlayerInTurn() { return inTurn; }
```

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

```
public class ShowUpdating {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to see Drawing updates",
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes reflected on the GUI");

    // Try to set the selection tool instead to see
    // completely free movement of figures, including the icon

    // editor.setTool( new SelectionTool(editor) );
  }
}
```

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

```
public class ShowUpdating {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to see Drawing updates",
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes reflected on the GUI");

    // Try to set the selection tool instead to see
    // completely free movement of figures, including the icon

    // editor.setTool( new SelectionTool(editor) );
  }
}
```

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

```
public class ShowUpdating {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to see
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes ref

    // Try to set the selection tool instead to see
    // completely free movement of figures, including the

    // editor.setTool( new SelectionTool(editor) );
  }
}
```

hotciv.visual.HotCivFactory4

```
/** Factory for visual testing of various SWEA template code */
class HotCivFactory4 implements Factory {
  private Game game;
  public HotCivFactory4(Game g) { game = g; }

  public DrawingView createDrawingView( DrawingEditor editor ) {
    DrawingView view =
      new MapView(editor, game);
    return view;
  }

  public Drawing createDrawing( DrawingEditor editor ) {
    return new CivDrawing( editor, game );
  }

  public JTextField createStatusField( DrawingEditor editor ) {
    JTextField f = new JTextField("SWEA template code");
    f.setEditable(false);
    return f;
  }
}
```

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

hotciv.visual.HotCivFactory4

hotciv.view.CivDrawing

```
public class ShowUpdating {
  public sta
    Game gam
    DrawingE
      new Mi
    editor.d
    editor.s
    editor.s
    // Try t
    // compl
    // edito
  }
}
```

```
/** Factory for visual testing of various SWEA template code */
class HotCivFactory4 implements Factory {
  private Game game;
  public HotCivFactory4(Game g) { game = g; }

  public DrawingView createDrawingView( Drawing
    DrawingView view =
      new MapView(editor, game);
    return view;
  }

  public Drawing createDrawing( DrawingEditor e
    return new CivDrawing( editor, game );
  }

  public JTextField createStatusField( DrawingE
    JTextField f = new JTextField("SWEA templat
    f.setEditable(false);
    return f;
  }
}
```

```
/** the Game instance that this CivDrawing is going to render units
 * from */
protected Game game;

public CivDrawing( DrawingEditor editor, Game game ) {
  super();
  this.delegate = new StandardDrawing();
  this.game = game;
  this.unitFigureMap = new HashMap<>();

  // register this unit drawing as listener to any game state
  // changes...
  game.addObserver(this);
  // ... and build up the set of figures associated with
  // units in the game.
  defineUnitMap();
  // and the set of 'icons' in the status panel
  defineIcons();
}
```

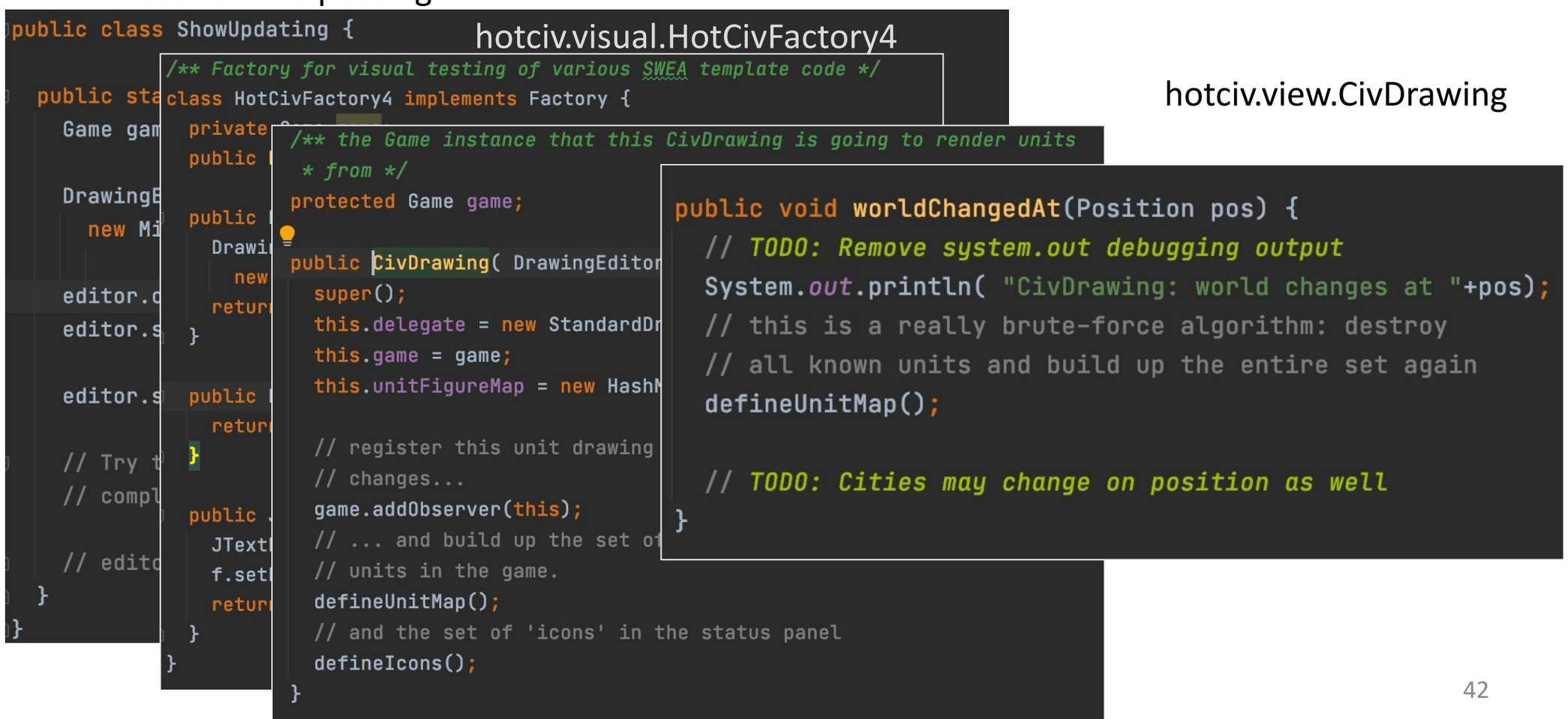
Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

hotciv.visual.ShowUpdating

hotciv.visual.HotCivFactory4

hotciv.view.CivDrawing



Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):

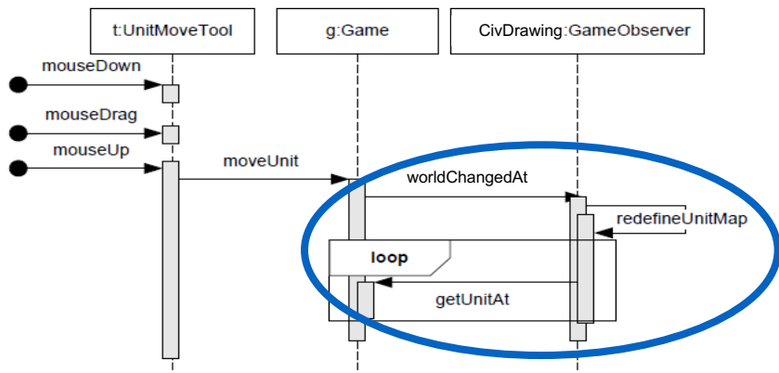
hotciv.visual.ShowUpdating

hotciv.visual.HotCivFactory4

hotciv.view.CivDrawing

Iteration 8: Observer (Update)

Use a StubGame (use provided code as a guide):



hotciv.visual.ShowUpdating

```
public class ShowUpdating {
  public sta
    Game gam
    DrawingE
      new Mi
    editor.d
    editor.s
    editor.s
    // Try t
    // compl
    // edito
  }
}
```

hotciv.visual.HotCivFactory4

```
/** Factory for visual testing of various SWEA template code */
class HotCivFactory4 implements Factory {
  private
  public
  public
    Drawi
      new
    retur
  }
}
```

```
/** the Game instance that this CivDrawing is going to render units
 * from */
protected Game game;

public CivDrawing( DrawingEditor editor, Game game
  super();
  this.delegate = new StandardDrawing();
```

```
public void worldChangedAt(Position pos) {
  // TODO: Remove system.out debugging output
  System.out.println( "CivDrawing: world chang
  // this is a really brute-force algorithm: d
  // all known units and build up the entire s
  defineUnitMap();

  // TODO: Cities may change on position as we
}
```

hotciv.view.CivDrawing

```
protected void defineUnitMap() {
  // ensure no units of the old list are accidental in
  // the selection!
  clearSelection();

  // remove all unit figures in this drawing
  removeAllUnitFigures();

  // iterate world, and create a unit figure for
  // each unit in the game world, as well as
  // create an association between the unit and
  // the unitFigure in 'unitFigureMap'.
  Position p;
  for ( int r = 0; r < GameConstants.WORLDSIZE; r++ ) {
    for ( int c = 0; c < GameConstants.WORLDSIZE; c++ ) {
      p = new Position(r,c);
      Unit unit = game.getUnitAt(p);
      if ( unit != null ) {
        String type = unit.getTypeString();
        // convert the unit's Position to (x,y) coordinates
```

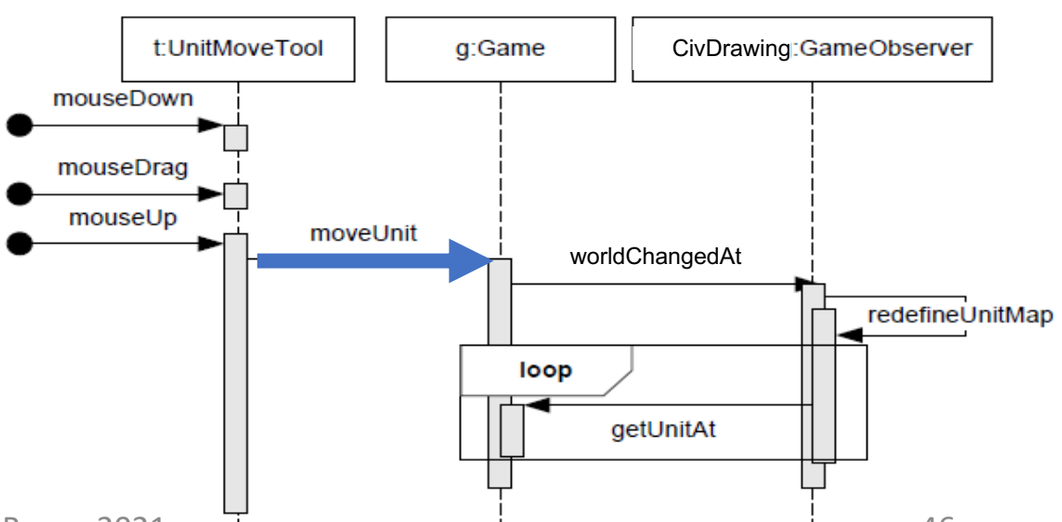
Iteration 8: Observer (Update)

- We have a StubGame ☑
- We have the CivDrawing we want to implement ☑
- We have made CivDrawing an observer on Game (our Stub) ☑
- We need a way to trigger the state changes for testing **TODO**

Iteration 8: Observer (Update)

- We have a StubGame ☑
- We have the CivDrawing we want to implement ☑
- We have made CivDrawing an observer on Game (our Stub) ☑
- We need a way to trigger the state changes for testing

TODO



Iteration 8: Observer (Update)

Trigger updates from Game:

hotciv.visual.ShowUpdating

```
public class ShowUpdating {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Click anywhere to see Drawing updates",
                               new HotCivFactory4(game) );
    editor.open();
    editor.setTool( new UpdateTool(editor, game) );

    editor.showStatus("Click anywhere to state changes reflected on the GUI");
```

Iteration 8: Observer (Update)

hotciv.visual.ShowUpdating

Trigger updates from Game:

hotciv.visual.ShowUpdating

```
editor.setTool( new UpdateTool(editor, game) );
```

```
class UpdateTool extends NullTool {
  private Game game;
  private DrawingEditor editor;
  public UpdateTool(DrawingEditor editor, Game game) {
    this.editor = editor;
    this.game = game;
  }
```

```
private int count = 0;
public void mouseDown(MouseEvent e, int x, int y) {
  switch(count) {
  case 0: {
    editor.showStatus( "State change: Moving archer to (1,1)" );
    game.moveUnit( new Position( r: 2, c: 0), new Position( r: 1, c: 1) );
    break;
  }
  case 1: {
    editor.showStatus( "State change: Moving archer to (2,2)" );
    game.moveUnit( new Position( r: 1, c: 1), new Position( r: 2, c: 2) );
    break;
  }
  case 2: {
    editor.showStatus( "State change: End of Turn (over to blue)" );
    game.endOfTurn();
    break;
  }
  case 3: {
    editor.showStatus( "State change: End of Turn (over to red)" );
    game.endOfTurn();
    break;
  }
  case 4: {
    editor.showStatus( "State change: Inspect Unit at (4,3)" );
    game.setTileFocus(new Position( r: 4, c: 3));
    break;
  }
```

Iteration 8: Observer (Update)

hotciv.visual.ShowUpdating

```
class UpdateTool extends NullTool {
  private Game game;
  private DrawingEditor editor;
  public UpdateTool(DrawingEditor editor, Game game) {
    this.editor = editor;
    this.game = game;
  }
```

Trigger updates from Game:

hotciv.visual.ShowUpdating

```
editor.setTool( new UpdateTool(editor, game) );
```

```
private int count = 0;
public void mouseDown(MouseEvent e, int x, int y) {
  switch(count) {
  case 0: {
    editor.showStatus( "State change: Moving archer to (1,1)" );
    game.moveUnit( new Position( r: 2, c: 0), new Position( r: 1, c: 1) );
    break;
  }
  case 1: {
    editor.showStatus( "State change: Moving archer to (2,2)" );
    game.moveUnit( new Position( r: 1, c: 1), new Position( r: 2, c: 2) );
    break;
  }
  case 2: {
    editor.showStatus( "State change: End of Turn (over to blue)" );
    game.endOfTurn();
    break;
  }
  case 3: {
    editor.showStatus( "State change: End of Turn (over to red)" );
    game.endOfTurn();
    break;
  }
  case 4: {
    editor.showStatus( "State change: Inspect Unit at (4,3)" );
    game.setTileFocus(new Position( r: 4, c: 3));
    break;
  }
```

First click

Second click

Third click

Iteration 8: Observer (Update)

TODO (You):

- TDD: Keep focus, take small steps
- Add a test:
  - Add another ‘click event’ to UpdateTool that forces a Game state change
- See it fail:
  - See the GUI ***not updating at all***
- Add (just enough) production code:
  - Develop further code in CivDrawing
- Until tests pass!

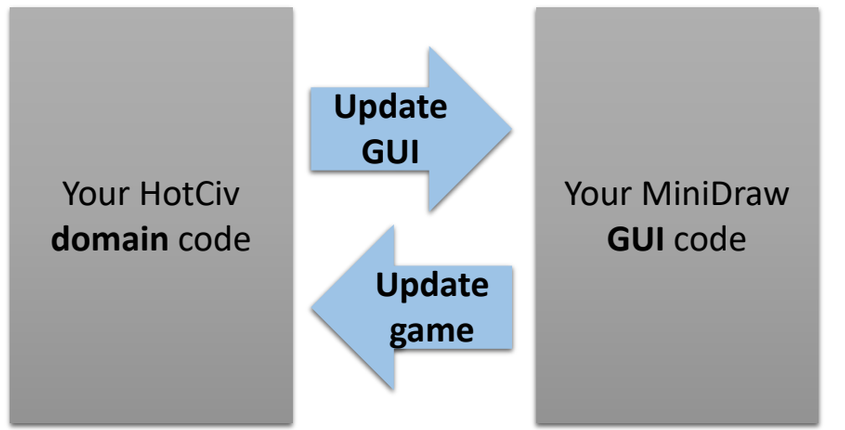
Iteration 8: Integration Part 3

- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).
- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.

Iteration 8: Tools

Focus:

- The GUI should change the state of the game
  - Dragging a unit translates to ‘moveUnit()’ call, etc.
- **GUI informs Domain**



Iteration 8: Tools

Focus:

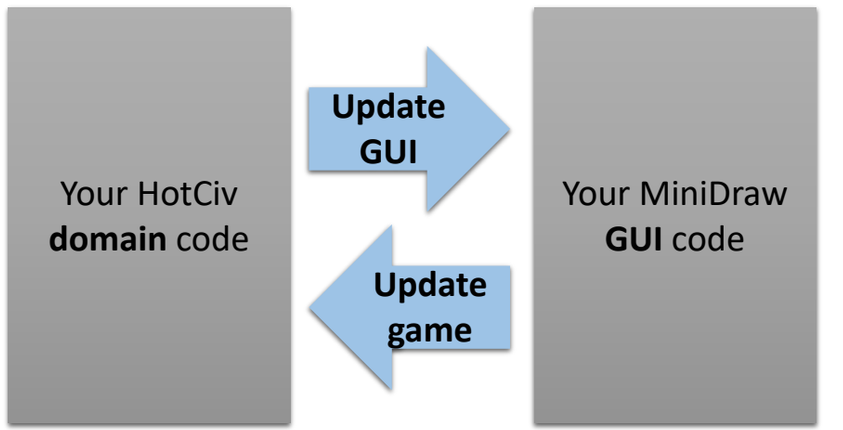
- The GUI should change the state of the game
  - Dragging a unit translates to 'moveUnit()' call, etc.
- **GUI informs Domain**

Testing **cannot** be automated:

- Use (manual) visual inspection

Use a StubGame:

- StubGame2 provides some (but not all) that is needed to test all tools



Observer pattern
[Domain *informs* GUI]

Facade pattern
[GUI *informs* Domain]

Iteration 8: Tools

Target: Move

→ Develop UnitMoveTool: drags a unit and calls moveUnit() in Game

hotciv.visual.ShowMove

```
public class ShowMove {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Move any unit using the mouse",
                               new HotCivFactory4(game) );
    editor.open();
    editor.showStatus("Move units to see Game's moveUnit method being called.");

    // TODO: Replace the setting of the tool with your UnitMoveTool implementation.
    editor.setTool( new SelectionTool(editor) );
  }
}
```

Iteration 8: Tools

Target: Move

→ Develop UnitMoveTool: drags a unit and calls moveUnit() in Game

hotciv.visual.ShowMove

```
public class ShowMove {

  public static void main(String[] args) {
    Game game = new StubGame2();

    DrawingEditor editor =
      new MiniDrawApplication( title: "Move any unit using the mouse",
                               new HotCivFactory4(game) );
    editor.open();
    editor.showStatus("Move units to see Game's moveUnit method being called.");

    // TODO: Replace the setting of the tool with your UnitMoveTool implementation.
    editor.setTool( new UnitMoveTool(editor, game) );
  }
}
```

Iteration 8: Tools

Target: Move

→ Develop UnitMoveTool: drags a unit and calls moveUnit() in Game

Tips:

See SelectionTool for mouse event handling

```
public void mouseDown(MouseEvent e, int x, int y) {
  Drawing model = editor().drawing();

  model.lock();

  draggedFigure = model.findFigure(e.getX(), e.getY());

  if (draggedFigure ≠ null) {
    fChild = createDragTracker(draggedFigure);
  } else {
    if (!e.isShiftDown()) {
      model.clearSelection();
    }
```

Use status messages

```
System.out.println("UnitMoveTool: Valid unit selected");
```

Iteration 8: Tools

One gradle target (task) per tool:

- **36.39: (Gradle target: move)** Develop a UnitMoveTool.
- **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
- **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
- **36.43: (Gradle target: action)** Develop an ActionTool.
- **36.44: (Gradle target: comp)** Develop a CompositionTool

Iteration 8: Tools

One gradle target (task) per tool:

- **36.39: (Gradle target: move)** Develop a UnitMoveTool.
- **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
- **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
- **36.43: (Gradle target: action)** Develop an ActionTool.
- **36.44: (Gradle target: comp)** Develop a CompositionTool

Each exercise has its own main program for visual testing (part of the provided code) – you are to fill in required code in each main function:

- ShowMove
- ShowSetFocus
- ShowEndOfTurn
- ShowAction
- ShowComposition

Iteration 8: Tools

Make tools **context sensitive**:

- Limit a tool's applicability to rectangles that define what is being clicked
- See hints in project iteration description
- See hotciv.view.GfxConstants

```
// === Constants that define positions for props on the gfx display
public final static int TURN_SHIELD_X = 559;
public final static int TURN_SHIELD_Y = 64;
public final static int AGE_TEXT_X = 535;
public final static int AGE_TEXT_Y = 23;

public final static int UNIT_SHIELD_X = 594;
public final static int UNIT_SHIELD_Y = 188;
public final static int UNIT_COUNT_X = 598;
public final static int UNIT_COUNT_Y = 256;

public static final int CITY_SHIELD_X = 595;
public static final int CITY_SHIELD_Y = 342;
public static final int WORKFORCEFOCUS_X = 590;
public static final int WORKFORCEFOCUS_Y = 444;
public static final int CITY_PRODUCTION_X = 595;
public static final int CITY_PRODUCTION_Y = 400;

public static final int REFRESH_BUTTON_X = 510;
public static final int REFRESH_BUTTON_Y = 472;
```

See MiniDraw tool behavior
for reference:

minidraw.standard.SelectionTool

minidraw.standard.handlers.DragTracker

etc.

Iteration 8: Tools

Make tools **context sensitive**:

- Limit a tool's applicability to rectangles that define what is being clicked
- See hints in project iteration description
- See hotciv.view.GfxConstants

How to decide which tool is active?

Exercise 36.44 Hint: Reference minidraw.standard.SelectionTool for compositional design

```
public class SelectionTool extends AbstractTool implements Tool {

    Sub tool to delegate to. The selection tool is in itself a state tool that may be in one of several states
    given by the sub tool. Class Invariant: fChild tool is never null
  protected Tool fChild;
```

```
draggedFigure = model.findFigure(e.getX(), e.getY());

if (draggedFigure ≠ null) {
  fChild = createDragTracker(draggedFigure);
} else {
  if (!e.isShiftDown()) {
    model.clearSelection();
  }
  fChild = createAreaTracker();
}
fChild.mouseDown(e, x, y);
```

Iteration 8: Integration Part 4

- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).
- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.

Iteration 8: Integration Part 4

Configure your MiniDraw GUI with your most advanced HotCiv game

- SemiCiv – combines advanced behavior from all variants

Play/test that it works!

Iteration 8: Summary

Use TDD!

- Define a test:
  - “MapView”: **OneStepTest:** draw the world…
  - “TestModelUpdate”: see game state changes reflected
  - …
  - Implement the MiniDraw role implementations to make it happen!
- Cannot automate all testing, but manual (visual) tests are better than no tests!

Iteration 8: Summary

Avoid Big Bang issues, **take small steps**

Use Test Stubs and Test Spies

- Some additional effort to code the test doubles/stubs/spies
- But, easier to spot defects and ensure that you **program to an interface**

Iteration 8: Summary

Week 1

- Merge the provided code **carefully**, in a **separate branch**, following instructions in Iteration 8.
- **36.37:** Implement and test the Observer pattern 'Subject' role behavior in your HotCiv framework code. Make the test cases part of your test suite.
- **36.38: (Gradle target: update)** Complete the implementation of the provided CivDrawing class such that all state changes in a Game are observed and reflected in proper GUI updates. (Hint: It is a bad idea to integrate with your real HotCiv Game implementation, instead extend the provided StubGame2.java or make another stub implementation to avoid difficult-to-test conditions. (Note that some 'observer related' code needs to be copied/duplicated from the Observer exercise above into your stub game).

Week 2

- Tools:
  - **36.39: (Gradle target: move)** Develop a UnitMoveTool.
  - **36.40: (Gradle target: setfocus)** Develop a SetFocusTool that sets the focus on a tile and updates the GUI with city and/or unit information in the status panel.
  - **36.42: (Gradle target: turn)** Develop an EndOfTurnTool.
  - **36.43: (Gradle target: action)** Develop an ActionTool.
  - **36.44: (Gradle target: comp)** Develop a CompositionTool
- **SemiCiv GUI (Gradle target: semi)** Develop a complete GUI based SemiCiv for system testing: Combine your developed SemiCiv variant from the previous iterations with your solutions to the previous exercises for GUI-HotCiv integration.