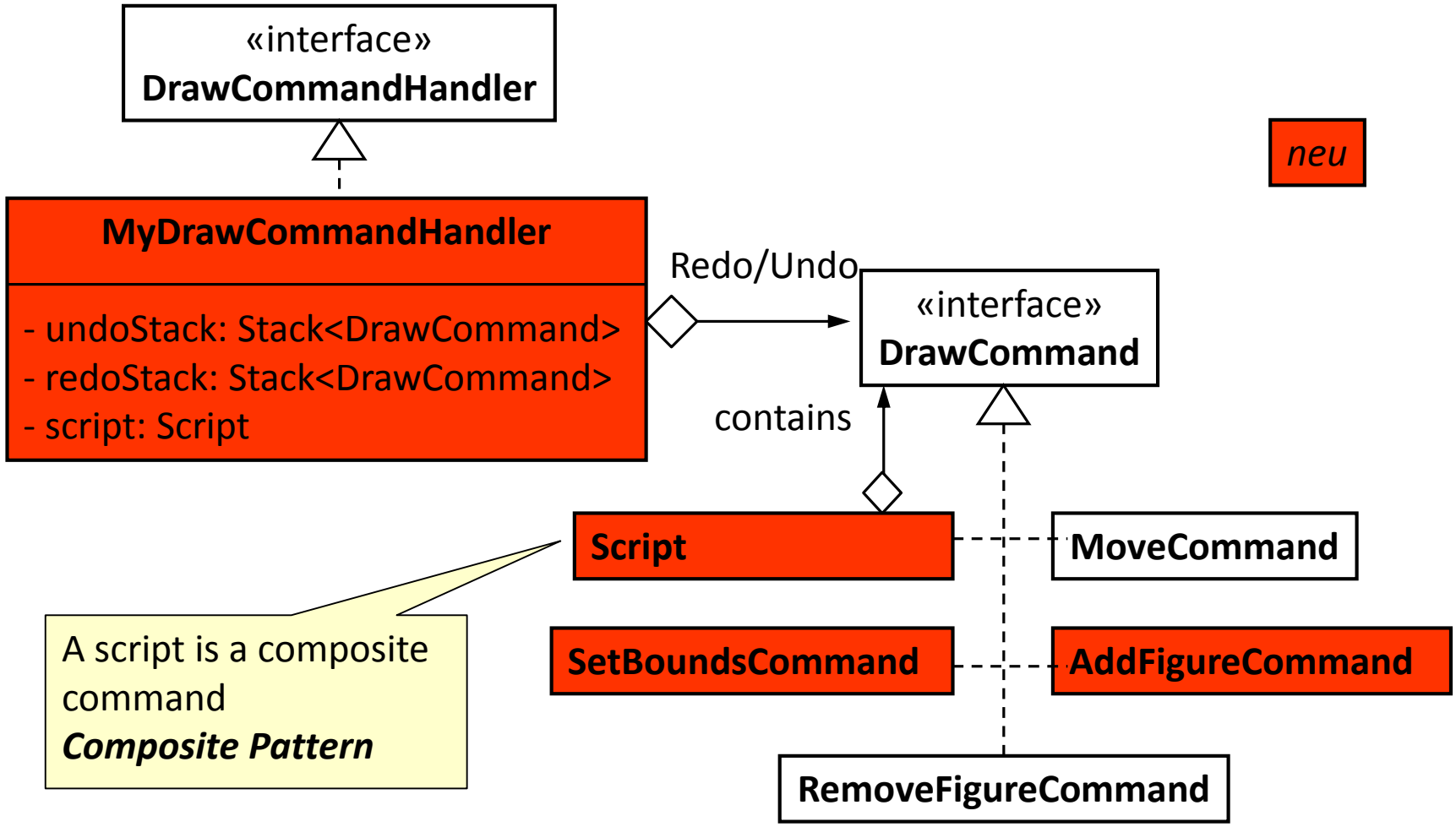
Assignment 10: Undo & Redo



MyDrawCommandHandler

```
public class MyDrawCommandHandler implements DrawCommandHandler {

  private Stack<DrawCommand> undoStack = new Stack<>();
  private Stack<DrawCommand> redoStack = new Stack<>();
  private Script script = null;

  public void addCommand(DrawCommand cmd) {
    redoStack.clear();
    if(script == null) {
      undoStack.push(cmd);
    } else {
      script.commands.add(cmd);
    }
  }

  …
```

Clears the redoStack whenever a new command is added

A new command is either added on the undo-stack or to an open script; this implementation does not support recursive scripts!

MyDrawCommandHandler

```
public void undo() {
  if(undoPossible()) {
    DrawCommand d = undoStack.pop();
    redoStack.push(d);
    d.undo();
  }
}

public void redo() {
  if (redoPossible()) {
    DrawCommand d = redoStack.pop();
    undoStack.push(d);
    d.redo();
  }
}

public boolean undoPossible() { return undoStack.size() > 0; }
public boolean redoPossible() { return redoStack.size() > 0; }
```

MyDrawCommandHandler: Scripts

```
public void beginScript() {
  if (script != null) throw new IllegalStateException();
  script = new Script();
}

public void endScript() {
  if (script == null) throw new IllegalStateException();
  Script s = script;
  script = null;
  if (s.commands.size() > 0) {
    if(s.commands.size() == 1) {
      addCommand(s.commands.get(0));
    } else {
      addCommand(s);
    }
  }
}
…
```

As the script is no longer null, new commands are added to the script

Script reference has to be cleared before script-command is added with `addCommand` (otherwise it would be added to itself)

Recursive scripts could be provided with a stack of open scripts

MyDrawCommandHandler: Scripts

```
  private static final class Script implements DrawCommand {
    private List<DrawCommand> commands = new LinkedList<>();

    public void redo() {
      ListIterator<DrawCommand> it = commands.listIterator();
      while (it.hasNext()) { it.next().redo(); }
    }

    public void undo() {
      int size = commands.size();
      ListIterator<DrawCommand> it = commands.listIterator(size);
      while (it.hasPrevious()) { it.previous().undo(); }
    }
  }

}
```

Undo has to traverse the list backwards

AddFigureCommand

```
public class AddFigureCommand implements DrawCommand {
  private final DrawModel model;
  private final Figure figure;

  public AddFigureCommand(DrawModel model, Figure figure) {
    this.model = model;
    this.figure = figure;
  }

  public void redo() {
    model.addFigure(figure);
  }

  public void undo() {
    model.removeFigure(figure);
  }

}
```

A new `AddFigureCommand` instance is added when a new figure has been created, e.g. in method `mouseUp` of the draw-tool:

```
context.getModel().
  getDrawCommandHandler().
    addCommand(
      new AddFigureCommand(
        context.getModel(), figure)
  );
```

SetBoundsCommand

```
public class SetBoundsCommand implements DrawCommand {
  private final Figure figure; // Figure on which setBounds was applied
  private final Point fromOrig, fromCorn, toOrig, toCorn;

  public SetBoundsCommand(Figure figure,
      Point fromOrig, Point fromCorn, Point toOrig, Point toCorn) {
    this.figure = figure;
    this.fromOrig = fromOrig;
    this.fromCorn = fromCorn;
    this.toOrig = toOrig;
    this.toCorn = toCorn;
  }

  public void redo() { figure.setBounds(toOrig, toCorn); }

  public void undo() { figure.setBounds(fromOrig, fromCorn); }

}
```

SetBoundsCommand

- **Where is the SetBoundsCommand instance created?**
  - In method `setBounds` of the figure implementations
    - setBounds is already called when a new figure is created, and these changes should not be undoable
  - In method `dragInteraction` of the handle implementations
    - "incremental" setBounds-commands are collected in a script, but only the last command would be necessary
  - In method `stopInteraction` of the handle implementations
    - `fromOrigin` and `fromCorner` have to be stored in method `startInteraction`

GroupFigureCommand

```
public class GroupFigureCommand implements DrawCommand {
  private final DrawModel model;
  private final FigureGroup group;
  private final boolean insertGroup;

  public GroupFigureCommand(DrawModel model, FigureGroup group,
                            boolean insertGroup) {
    this.model = model; this.group = group;
    this.insertGroup = insertGroup;
  }

  public void redo() {
    if (insertGroup) { insertGroup(); } else { removeGroup(); }
  }

  public void undo() {
    if (insertGroup) { removeGroup(); } else { insertGroup(); }
  }
```

Indicates whether the current command instance is a group or an ungroup operation

GroupFigureCommand

```
  private void insertGroup() {
    for (Figure f : group.getFigureParts()) {
      model.removeFigure(f);
    }
    model.addFigure((Figure) group);
  }

  private void removeGroup() {
    model.removeFigure((Figure) group);
    for (Figure f : group.getFigureParts()) {
      model.addFigure(f);
    }
  }


}
```

Problem: the figures are added at the end of the figure list and are therefore probably not at the same position as before the grouping operation!