COMP 401
COMMAND OBJECTS AND UNDO

Instructor: Prasun Dewan
PREREQUISITES

- Animation Threads Commands
TOPICS

- Command Object
  - Object representing an action invocation such as “Do your homework”.

- Threads
  - Support non blocking action invocation.

- Undo/Redo
  - Supports undoable/re-doable commands (action invocations)
UNDO: INITIAL STATE

Initial State
Undo Initial State
CANNOT ALWAYS UNDO

If no command executed, undo does nothing
**Redo Initial State**

If no command executed, undo does nothing
If no command executed, redo does nothing
CHANGE HEIGHT

Execute change height command.
Execute change weight command.

Height and bmi change.
Undo
Weight and BMI both undone to restore state before setWeight() call
Redo

Redo last undone command
REDO

Last undone command reexecuted
LAST COMMAND UNDO

Redo after a redo.
CANNOT ALWAYS REDO

No undo commands to redo
Undo Implementation

Weight and BMI both undone to restore state before `setWeight()` call.

`setWeight()` called with old weight.

Undoable command object remembers method and its parameters.

In the application could have multiple spreadsheets, points, ... all sharing one undo history.

Special global undoer keeps track of command history.

`setWeight` in `undoable` creates command and gives it to undoer.
UNDO IN BMI

ABMISpreadsheet

AHistoryUndoer

AnUndoableBMISpreadsheet

ASetWeight(Height)Command
public interface Undoer {
    public void undo();
    public void execute(Command command);
    public void redo();
}
HISTORY UNDOER

Command 1
Command 2
Command 3
Command 4
Command 5
Command 6
Command 7
Command 8
Command 9
UNDO

Command 1
Command 2
Command 3
Command 4
Command 5
Command 6
Command 7
Command 8
Command 9
Command 1
Command 2
Command 3
Command 4
Command 5
Command 6
Command 7
Command 8
Command 9
EXECUTE

- Command 1
- Command 2
- Command 3
- Command 4
- Command 5
- Command 6
- Command 7
- Command 8
- Command 9
EXECUTE

- Command 1
- Command 2
- Command 3
- Command 4
- Command 5
Execute

Command 1
Command 2
Command 3
Command 4
Command 5
Command 10
```java
public class HistoryUndoer implements Undoer {
    List<Command> historyList = new ArrayList();
    int nextCommandIndex = 0;
    public void execute (Command c) {
        while (nextCommandIndex < historyList.size()) {
            historyList.remove(nextCommandIndex); // clear redo chain
        }
        c.execute();
        historyList.add(c);
        nextCommandIndex++;
    }
    public void undo() {
        if (nextCommandIndex == 0) return;
        nextCommandIndex--;
        Command c = historyList.get(nextCommandIndex);
        c.undo();
    }
    public void redo() {
        if (nextCommandIndex == historyList.size()) return;
        Command c = historyList.get(nextCommandIndex);
        c.execute();
        nextCommandIndex++;
    }
}
```
public interface Command {
    public void execute();
    public void undo();
}
public class ASetWeightCommand implements Command {
BMISpreadsheet bmiSpreadsheet;
  double oldWeight;
  double weight;
public ASetWeightCommand (BMISpreadsheet theBMISpreadsheet, double theWeight) {
    bmiSpreadsheet = theBMISpreadsheet;
    weight = theWeight;
    oldWeight = bmiSpreadsheet.getWeight();
}
public void execute() {bmiSpreadsheet.setWeight(weight);}
public void undo() {bmiSpreadsheet.setWeight(oldWeight);}
}
**ASetHeightCommand**

```java
public class ASetHeightCommand implements Command {
    BMISpreadsheet bmiSpreadsheet;
    double oldHeight;
    double height;

    public ASetHeightCommand (BMISpreadsheet theBMISpreadsheet, double theHeight) {
        bmiSpreadsheet = theBMISpreadsheet;
        height = theHeight;
        oldHeight = bmiSpreadsheet.getHeight();
    }

    public void execute() {bmiSpreadsheet.setHeight(height);}
    public void undo() {bmiSpreadsheet.setHeight(oldHeight);}
}
```

Reflection could allow these two command objects to be combined.
**Undoable BMISpreadsheet**

```java
public interface UndoableBMISpreadsheet extends BMISpreadsheet{
    public void redo();
    public void undo();
}
```

Usually user-invokable undo/redo methods would be provided by a global application object for all objects in the application.

Here there is only one application object so undo/redo in UndoableBMISpreadsheet

The interface of AnUndoableSpreadsheet and BMiSpreadsheet would be same if global object
public class AnUndoableBMISpreadsheet implements UndoableBMISpreadsheet {
    BMISpreadsheet bmiSpreadsheet;
    Undoer undoer;

    public AnUndoableBMISpreadsheet (BMISpreadsheet theBMISpreadsheet, Undoer theUndoer) {
        bmiSpreadsheet = theBMISpreadsheet;
        undoer = theUndoer;
    }

    public double getBMI () {
        return bmiSpreadsheet.getBMI();
    }

    public double getHeight () {
        return bmiSpreadsheet.getHeight();
    }

    public double getWeight () {
        return bmiSpreadsheet.getWeight();
    }
}
public void setHeight(double theHeight) {
    undoer.execute(new ASetHeightCommand(bmiSpreadsheet, theHeight));
}

public void setWeight(double theWeight) {
    undoer.execute(new ASetWeightCommand(bmiSpreadsheet, theWeight));
}

public void undo() {undoer.undo();}
public void redo() {undoer.redo();}
public void setHeight(double theHeight) {
    undoer.execute(new ASetHeightCommand(this, theHeight));
}

public void setWeight(double theWeight) {
    undoer.execute(new ASetWeightCommand(this, theWeight));
}

public void undo() {undoer.undo();}
public void redo() {undoer.redo();}
**Undoable BMISpreadsheet (review)**

```java
public interface UndoableBMISpreadsheet extends BMISpreadsheet{
    public void redo();
    public void undo();
}
```

Usually user-invokable undo/redo methods would be provided by a global application object for all objects in the application.

Here there is only one application object so undo/redo in UndoableBMISpreadsheet

The interface of AnUndoableSpreadsheet and BMiSpreadsheet would be same if global object


public class AnUndoableBMISpreadsheet implements UndoableBMISpreadsheet {

    BMISpreadsheet bmiSpreadsheet;
    Undoer undoer;

    public AnUndoableBMISpreadsheet (BMISpreadsheet theBMISpreadsheet, Undoer theUndoer) {
        bmiSpreadsheet = theBMISpreadsheet;
        undoer = theUndoer;
    }

    public double getBMI() {
        return bmiSpreadsheet.getBMI();
    }

    public double getHeight() {
        return bmiSpreadsheet.getHeight();
    }

    public double getWeight() {
        return bmiSpreadsheet.getWeight();
    }
}
UNDOABLE BMISpreadsheet (review)

```java
public void setHeight(double theHeight) {
    undoer.execute(new ASetHeightCommand(bmiSpreadsheet, theHeight));
}

public void setWeight(double theWeight) {
    undoer.execute(new ASetWeightCommand(bmiSpreadsheet, theWeight));
}

public void undo() {
    undoer.undo();
}

public void redo() {
    undoer.redo();
}
```
**UNDOABLE BMISpreadsheet (review)**

```java
public void setHeight(double theHeight) {
    undoer.execute(new ASetHeightCommand(this, theHeight));
}

public void setWeight(double theWeight) {
    undoer.execute(new ASetWeightCommand(this, theWeight));
}

public void undo() {
    undoer.undo();
}

public void redo() {
    undoer.redo();
}
```

*This would cause infinite recursion*
INTERMEDIARY DELEGATING CLASS

client → AnUndoableBMISpreadsheet → ABMISpreadsheet
Adapter is a class that sits between a client and adaptee class much like an adapter sits between two objects that need to interact with each other.

- Methods called in adaptee through adapter.
- Degree of adaptation undefined.
  - Assumed no extra functionality offered but some may be removed.
- Methods offered to client
  - Adapted name
  - Adapted parameters.
Proxy is a class that sits between a client and subject class, offering the same interface.
Proxy is a stand-in for real subject.
Methods called in subject through proxy methods.
A proxy method does not change the behavior of subject method.
A proxy can add functionality.
Like a regulated power supply, or one with a special fuse.
Proxies in Everyday Apps

- Proxies adding support for:
  - Logging
  - Collaboration
  - Cache data
  - Redirect to nearest server
  - Access control
  - Assertions
  - Undo/redo
**UNDO PATTERN**

**ABMISpreadsheet**

**AnUndoableBMISpreadsheet**

**ASetWeight(Height)Command**

**AHistoryUndoer**

**Defines executable methods**

**Executer**

**Undo and execute undoable method(s)**

**Undoable Command(s)**

**Undoable**

**Provides proxy undoable methods that instantiate commands, and interacts with undoer**

**Undoer**

**Chooses undo/redo command and invokes undo/execute on it**
Changing Undoable

AnotherBMISpreadsheet → ASetWeight(Height)Command

AnUndoableBMISpreadsheet → AHistoryUndoer

Executer → Undoable Command(s)

Undoable → Undoer
CHANGING UNDOER

AnotherBMISpreadsheet

AnUndoableBMISpreadsheet

ASetWeight(Height)Command

ALastCommandUndoer

Executer

Undoable

Undoable Command(s)

Undoer
CHANGING EVERYTHING BUT UNDOER

ACartesianPoint

AnUndoablePoint

ALastCommandUndoer

ASetX(Y)Command

Undoable Command(s)

Undoer

Undoable

Executer
Global User Interface Undoer

All objects in a UI share a single Undoer
**Undo Pattern**

![Diagram showing the relationships between ABMISpreadsheet, ASetWeight(Height)Command, AnUndoableBMISpreadsheet, ALastCommandUndoer, Executer, Undoable Command(s), Undoable, and Undoer.]

- ABMISpreadsheet
- ASetWeight(Height)Command
- AnUndoableBMISpreadsheet
- ALastCommandUndoer
- Executer
- Undoable Command(s)
- Undoable
- Undoer
Executing an Undoable Method

```java
public void setWeight(double theWeight) {
    undoer.execute(new ASetWeightCommand(bmiSpreadsheet, theWeight));
}
```

Delegating setWeight() creates command and asks undoer to execute it.
public void execute (Command c) {
    if (nextCommandIndex != historyList.size()) {
        historyList.clear(); //ignore remaining undone commands
        nextCommandIndex = 0;
    }
    c.execute();
    historyList.add(c);
    nextCommandIndex++;
}

Delegating setWeight() creates command and asks undoer to execute it.

Undoer calls execute() in command.
EXECUTING AN UNDOABLE METHOD

public class ASetWeightCommand implements Command {
    BMISpreadsheet bmiSpreadsheet;
    double oldWeight;
    double weight;

    public ASetWeightCommand (BMISpreadsheet theBMISpreadsheet, double theWeight) {
        bmiSpreadsheet = theBMISpreadsheet;
        weight = theWeight;
        oldWeight = bmiSpreadsheet.getWeight();
    }

    public void execute() {bmiSpreadsheet.setWeight(weight);}
    public void undo() {bmiSpreadsheet.setWeight(oldWeight);}
}

Delegating setWeight() creates command and asks undoer to execute it.

Undoer calls execute() in command.

Command asks delegate to invoke() setWeight() with constructor parameter value 66.0.
Delegating undo asks undoer to execute undo.
Delegating undo asks undoer to execute undo.

`undoer` finds command object of last command and calls undo method of command object.

```
public void undo() {
    if (nextCommandIndex == 0) return;
    nextCommandIndex--;
    Command c = historyList.get(nextCommandIndex);
    c.undo();
}
```
public class ASetWeightCommand implements Command {
    BMISpreadsheet bmiSpreadsheet;
    double oldWeight;
    double weight;
    public ASetWeightCommand (BMISpreadsheet theBMISpreadsheet, double theWeight) {
        bmiSpreadsheet = theBMISpreadsheet;
        weight = theWeight;
        oldWeight = bmiSpreadsheet.getWeight();
    }
    public void execute() { bmiSpreadsheet.setWeight(weight);}
    public void undo() { bmiSpreadsheet.setWeight(oldWeight);}
}

Delegating undo asks undoer to execute undo.

Undoer finds command object of last command and calls undo method of command object.

Undo method of command object calls setWeight() method of delegate with old value of weight: 75.0
**Undo Effect**

command undone.
Delegating redo asks undoer to execute redo.
Delegating redo asks undoer to execute redo.

Undoer finds last undone command object and calls execute method of command object.
public class ASetWeightCommand implements Command {
    BMISpreadsheet bmiSpreadsheet;
    double oldWeight;
    double weight;
    public ASetWeightCommand (BMISpreadsheet theBMISpreadsheet, double theWeight) {
        bmiSpreadsheet = theBMISpreadsheet;
        weight = theWeight;
        oldWeight = bmiSpreadsheet.getWeight();
    }
    public void execute() {bmiSpreadsheet.setWeight(weight);}
    public void undo() {bmiSpreadsheet.setWeight(oldWeight);}
}

execute redo.

Undoer finds last undone command object and calls redo method of command object.

Redo method of command object calls setWeight() method of delegate with its constructor parameter value : 66.0

Execute method of same command object executed multiple times!
redo effect

Last undone command reexecuted
EXTRA SLIDES
INHERITANCE-BASED PROXY PATTERN

ABMISpreadsheet implements BMISpreadsheet

IS-A

AnUndoableBMISpreadsheet implements Subject Class

Proxy Class implements Subject Interface

Subject Class is-a Subject Interface
DELEGATION-BASED PROXY PATTERN

ABMISpreadsheet implements BMISpreadsheet

HAS-A

AnUndoableBMISpreadsheet

HAS-A

Subject Class

implements

Subject Interface

WORKS FOR ALL IMPLEMENTATIONS OF SUBJECT INTERFACE (E.G. WEB SERVER)
INHERITANCE-BASED PROXY PATTERN

- Adding proxy functionality to subject class
- Inheriting Proxy
- Delegating Proxy

Ease of coding
Modularity Distribution
DELEGATION BASED COMMANDS

AHistoryUndoer implements Undoer

HAS-A

ASetWeightCommand implements Command

HAS-A

Command Invoker implements Command Invoker Interface

HAS-A

Command Class implements Command Interface
Delegation Based Commands

- Thread
  - HAS-A
  - AShuttleAnimationCommand
  - Runnable
    - implements
  - Command Invoker
    - implements
  - Command Class
    - Command Invoker Interface
      - HAS-A
      - implements
  - Command Interface
    - implements
INHERITANCE BASED COMMANDS

AShuttleAnimationCommand

Command Invoker

Command Class

Runnable

Thread

IS-A

implements

Command and Command Invoker Interface

Old but convenient and inflexible way of implementing threads
EXTRA SLIDES
COMMAND OBJECTS IN EVERYDAY APPS

- Thread: Runnables
- Undo/Redo: Undoable Command Object
ANIMATING VS. UPDATING CLASSES

In general, a method that performs the animation steps and a method that changes the value of some animating property may be in different classes:

- AnAnimatingShuttleLocation
- ALabel
Objects include methods and data

Java does not allow method parameters

Method Parameters in Java

Command Object

Method 1

Variable 1

Variable 2

Method 2

Parameter N
**COMMAND OBJECT**

When a method m1 wants to pass method m2 to method m3, it passed a command object for the method m2.

When method m3 wants to call m2 on object o, it passes:

- Parameterless method that calls method to be called.
- Constructor takes parameters of method to be called and target.
- Object on which method is to be called

Parameter 1
Parameter 2
Parameter N
**Method Parameters in Java**

Java does not allow method parameters

Objects include methods and data

Instead of passing a method, pass a command object

Command object = method + parameters
**Undoer BMISpreadsheet Separation**

- Can use BMISpreadsheet with different undoer.
- Can use undoer with different object.
- Can use undoer with multiple objects in a single user interface.
Proxy is a class that sits between a client and subject class.
Proxy is a stand-in for real subject.
Methods called in subject through proxy methods,
A proxy method does not change the behavior of subject method
A proxy adds subject-independent functionality – which is independent of specific subject.
- The interface of functionality is independent of subject interface (undo/redo)
- The implementation may not be (required subject-specific commands)
Proxy is a class that sits between a client and subject class.
Proxy is a stand-in for real subject.
Methods called in subject through proxy methods,
A proxy method does not change the behavior of subject method
A proxy adds functionality – which is independent of specific subject.
  • The interface of functionality is independent of subject interface (undo/redo)
  • The implementation may not be (required subject-specific commands)
Undoer/Undoable Separation

- ALastCommandUndoer implements Undoer
- AnUndoableBMISpreadsheet implements BMISpreadsheet
- Undoer Class implements Undoer Interface
- Undoable Class implements Undoable Interface
CHANGING THE UNDOER

AHistoryUndoer implements Undoer

HAS-A

AnUndoableBMISpreadsheet implements BMISpreadsheet

Undoer Class 2 implements Undoer Interface

HAS-A

Undoable Class implements Undoable Interface
Changing the Undoable

AHistoryUndoer implements Undoer

HAS-A

AnUndoableCartesianPoint implements Point

Undoer Class implements Undoer Interface

HAS-A

Undoable Class 2 implements Undoable Interface 2
Multiple Undoables Per Undoer

AHistoryUndoer implements Undoer

HAS-A

AnUndoablecartesianPoint HAS-A AnUndoableBMISpreadsheet

Undoer Class implements Undoer Interface

HAS-A

Undoable Class 2 HAS-A Undoable Class

All Objects in a UI share a single Undoer