CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2010
Doc 22 Assignment 3 Comments
29 Apr 2010

Names

public class Originator { }

public class Proxy {}

public class CareTaker { }

Command Issue

```
public class AddItemCommand extends Command {
    public AddItemCommand(Inventory orginator, String name, int quantity, double
    price) {
        blah
    }


    public void execute() {
        blah
    }
```

No inherited methods

Inner Classes

```
public class Foo {
    public void stuff() {
    }

    public void moreStuff() {
    }

    private static class Memento {
        protected void setState(blah) {
            blah
        }

        protected HashMap<blah> getState(blah) {
        }

    public void evenMoreStuff() {
        blah
    }
}
```

Rerepeat

```
public boolean addCopies(blah) {
    Command add = new AddCopiesCommand(blah);
    add.execute();
    try {
        objOutStr.writeObject(add);
        return true;
    } catch (Exception e ) {
        return false;
    }
}

public boolean sellMovie(blah) {
    Command sell = new sellMovieCommand(blah);
    sell.execute();
    try {
        objOutStr.writeObject(sell);
        return true;
    } catch (Exception e ) {
        return false;
    }
}
```

Memento

```
public class Inventory extends InventoryInterface {

    public Object getMemento () {
        return new InventoryMemento(movieTable);
    }

    public void restoreMemento() {
        Object oldState;
        ObjectInputStream objInStr =
            new ObjectInputStream(new FineInputStream("memento.obj");
        Object readMemento = objInStr.readObject();
        oldState = readMemento;
        objInStr.close();
        Hashtable<Integer, Movie> memento = (Hashtable<Integer, Movie>) oldState;
        this.inventoryTable = memento;
        Movie.movieId.incrementAndGet();
}
```

try-catch omitted to save space

Memento.txt

```
public Memento readMemento() {
    ObjectInputStream objInStr =
        new ObjectInputStream(new FineInputStream("memento.txt");
    Object Memento = objInStr.readObject();
    return (Hashtable<Integer, Movie>) Memento;
}
```

Memento File

How many class/methods access file name?

How many places is the file name hard coded?

Did you remember to make a copy of the existing file before you saved memento?

Did you remember check for the copy of the memento file when recovering?

Memento

```
public class Inventory extends InventoryInterface {

    public Object getMemento () {
        InventoryMemento currentState = new InventoryMemento(movieTable);
        code to save memento to file
        return currentState;
    }

    public void restoreMemento() {
        code to read and restore memento from file
    }
```

```
public class InventoryProxy implements blah {
    private in sizeOfList;
    private boolean flag = false;
    private Object[] args = new Object[4];
    private Command cmd;
    private Inventory target = new Inventory();
    private int movieQuantity;
    private int sameMovieQuantity;
    private List<Command> commandList;

    blah
}
```

State

```
public class InventoryProxy implements blah {
    private in sizeOfList;
    private boolean flag = false;
    private Object[] args = new Object[4];
    private Command cmd;
    private Inventory target = new Inventory();

    blah
}
```

```
public Object findPrice(String movieName, int id) {
    Object price=0;
    Movie dvd;

    if(id>0 && (movieName!=null || movieName == null)) {
        if (idAsKey.containsKey(id)) {
            return idAsKey.get(id).getPrice():
        } else {
            System.out.println("Movie with id " + id + "  not available");
            return price;
        }
    }
    if (id <0 && movieName != null ) {
        if (namesAsKey.containsKey(movieName)) {
            return nameAsKey.get(movieName).getPrice():
        } else {
            System.out.println("Movie with name " + movieName + "  not available");
            return price;
        }
    } else {
        System.out.println("Please enter valid name or id");
    }
}
```

What?

Shorter Version

```
public Object findPrice(String movieName, int id) {
    if (movieName != null) return findPrice(movieName);
    if (id > 0) return findPrice(id);
    return INVALID_MOVIE_IDENTIFIER;
}

private Object findPrice(String movieName) {
    if (!namesAsKey.containsKey(movie)) return MOVIE_NOT_FOUND;
    return nameAsKey.get(movieName).getPrice();
}

private Object findPrice(int id) {
    if (! idAsKey.containsKey(id))) return MOVIE_NOT_FOUND;
    return idAsKey.get(id).getPrice();
}
```

idAsKey.get(id).getPrice();

Nesting method calls is considerd poor style

Have no idea what you are interacting with

Better to do it in steps

```
Movie requestedMovie = idAsKey.get(id);
return requestedMovie.getPrice();
```

What?

```
public Object findPrice(String movieName, int id) {
Object price=0;
        Movie dvd;

    if(id>0 && (movieName!=null || movieName == null)) {
      if (idAsKey.containsKey(id)) {
                return idAsKey.get(id).getPrice():
       } else {
          System.out.println("Movie with id " + id + "  not available");
       return price;
      }
     }
     if (id <0 && movieName != null ) {
                if (namesAsKey.containsKey(movieName)) {
            return nameAsKey.get(movieName).getPrice():
       } else {
          System.out.println("Movie with name " + movieName + "  not available");
           return price;
       }
       } else {
System.out.println("Please enter valid name or id");
     }
}
```

In InventoryProxy Class

```
public void addCopy(int movidId, int quantity) throws MovieNotFoundException {
    try {
        Class[] argumentTypes = { Integer.TYPE, Integer.TYPE };
        Object[] arguments = {movieId, quantity };
        Command command = createCommand("addCopy", argumentTypes, arguments);
        command.execute(realSubject);
        commandSerializer.saveCommandToFile(command);
    } catch {Exception e ) {
        e.printStackTrace();
    }
}
```

In InventoryProxy Class

```
public void addMovie(String movieName, double price, int quantity)
                    throws MovieNotFoundException {
    try {
        Class[] argumentTypes = { String.class, Integer.TYPE, Integer.TYPE };
        Object[] arguments = {movieName, price, quantity };
        Command command = createCommand("addMovie", argumentTypes, arguments);
        command.execute(realSubject);
        commandSerializer.saveCommandToFile(command);
    } catch {Exception e ) {
        e.printStackTrace();
    }
}
```

In InventoryProxy Class

```
private Command createCommand(String name, Class[] types, Object[] arguments)
                    throws InvocationTargetException, etc {
    Command command = null;
    try {
        command = new Command(name, types, arguments);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return command;
}
```

Use Constructor methods

```
public class Command {

    public static Command addCommand(int movidId, int quantity) {
        try {
            Class[] types = { Integer.TYPE, Integer.TYPE };
            Object[] arguments = {movieId, quantity };
            return new Command(addCopy, types, arguments);
        } catch {Exception e ) {
            e.printStackTrace();
        }
        return null;
}
```

New addCopy

```
public void addCopy(int movidId, int quantity) throws MovieNotFoundException {
    try {
        Command command = Command.addCommand(movidId, quantity);
        command.execute(realSubject);
        commandSerializer.saveCommandToFile(command);
    } catch {Exception e ) {
        e.printStackTrace();
    }
}
```