Matt Gorelik

Refactoring

#1) I used the *Extract Method* refactoring for this one because of the duplication of very similar routines.

**class** Piece **extends** JPanel

{

/\* -- Other code -- \*/

**public** **void** paintComponent(Graphics g)

{

drawWhiteOvalOnColoredRect(0,Color.red, g);

drawWhiteOvalOnColoredRect(getHeight()/2, Color.blue,g);

}

//For this method, this refactoring works. If we wanted to allow for more parameters to be modified (ie, multiple color options, an option to change x position, etc.) we could extend this method, but in this example the oval was always white.

**public** **void** drawWhiteOvalOnColoredRect(**int** height, Color rectColor, Graphics g)

{

g.setColor(rectColor);

g.fillRect(getWidth()/4,height,getWidth()/2, getHeight()/2);

g.setColor(Color.white);

g.drawOval(getWidth()/4,height, getWidth()/2, getHeight()/2);

}

#2) I used the *Introduce Parameter Object* refactoring for this one because the parameter list was waaay too long for each of these methods. It could be much simpler with the introduction of a Point, Line, and Box class (better yet, an Intersectable interface which Line and Box both implement, but it’s not necessary).

**class** CollisionDetector

{

/\*\* Checks whether two line segments intersect \*/

**public** **static** **boolean** lineLineIntersect(Line l1, Line l2) {

/\* -- Code here -- \*/

}

/\*\* Checks whether a line segment intersects with a rectangular box \*/

**public** **static** **boolean** lineBoxIntersect(Line l, Box b) {

/\* -- Code here -- \*/

}

}

**class** Point {

**int** x;

**int** y;

}

**class** Box {

Point corner1;

Point corner2;

}

**class** Line {

Point one;

Point two;

}

#3) For this, I tried to replace conditional logic with polymorphism. I didn’t quite have time to finish, but here is where I was going. Of course, I also need to implement the Normal and Closed account states, but I wanted an AccountState interface with the 4 different states being implemented as singleton instances of the classes for each type of state. I then have the deposit, withdraw, and extractFee logic within these instances of the respective classes to avoid the excessive use of conditional logic.

**interface** AccountState {

**public** **void** deposit(Account act, **int** money);

**public** **void** withdraw(Account act, **int** money);

**public** **void** extractFee(Account act);

}

**class** Account

{

**public** **int** balance;

**public** AccountState state;

**public** **static** **final** AccountState *NORMAL* = **new** Normal();

**public** **static** **final** AccountState *UNDERFUNDED* = **new** Underfunded();

**public** **static** **final** AccountState *OVERDRAWN* = **new** Overdrawn();

**public** **static** **final** AccountState *CLOSED* = **new** Closed();

}

**class** Underfunded **implements** AccountState {

**public** **void** deposit(Account act, **int** money)

{

act.balance += money;

**if** (act.balance >= 500)

{

generateAuditTrial(**this**,act.state,Account.*NORMAL*);

act.state = Account.*NORMAL*;

}

}

**public** **void** withdraw(Account act, **int** money)

{

act.balance -= money;

**if** (act.balance < 0) {

generateAuditTrial(**this**, act.state, Account.*OVERDRAWN*);

act.state = Account.*OVERDRAWN*;

}

**else** **if** (act.balance < 500) {

generateAuditTrial(**this**, act.state, Account.*UNDERFUNDED*);

act.state = Account.*UNDERFUNDED*;

}

}

**public** **void** extractFee(Account act)

{

act.balance -= getCurrentUnderfundedFee();

}

**private** **int** getCurrentUnderfundedFee() {

// We need this somewhere; it could also be a stored variable, it doesn't really matter

**return** 0;

}

**private** **void** generateAuditTrial(Underfunded underfunded,

AccountState state, AccountState normal) {

// We are not sure what the AuditTrial really does

}

}

**class** Overdrawn **implements** AccountState{

**public** **void** deposit(Account act, **int** money)

{

act.balance += money;

**if** (act.balance >= 500)

{

generateAuditTrial(**this**, act.state, Account.*NORMAL*);

act.state = Account.*NORMAL*;

}

**if** (act.balance >= 0)

{

generateAuditTrial(**this**, act.state, Account.*UNDERFUNDED*);

act.state = Account.*UNDERFUNDED*;

}

}

**public** **void** withdraw(Account act, **int** money)

{

**throw** **new** Exception("Throw new underfunded exception");

}

**public** **void** extractFee(Account act)

{

accountBalance -= getCurrentOverdrawnFee();

}

**private** **int** getCurrentOverdrawnFee() {

// **TODO** Auto-generated method stub

**return** 0;

}

}

#4) For this one, I made a lot of inline adjustments, as well as removing some unnecessary conditional logic.

**class** Application

{

**public** Application() {

**boolean** condition = ... ; /\* Some runtime condition \*/

**if** (condition) {

UserConfiguration usrConfig = **new** UserConfiguration();

copyFile(usrConfig.getUserWorkspaceDirectory() + "data.txt", usrConfig.getUserTempDirectory() + "data.txt");

} **else** {

DefaultConfiguration defConfig = new DefaultConfiguration();

copyFile(defConfig.getDefaultRootDirectory() + "/.workspace" + "data.txt", "/tmp" + "data.txt");

}

}

/\* -- Other code -- \*/

}