**Refactoring Example 1**

StockManager.java class

GitHub Entry : 12/10/2011 08:07pm

Old code iterated through the 'stockNames' collection twice, as it had to be populated before being sorted to be used as a reference for alphabetical listing. The method was refactored such that the collection was populated at the moment in time when each portfolio item is added, so that the 'getSummary' method only has to sort the list and then iterate it once.

Refactored code reduces the time complexity of the 'getSummary' function as well as making the code logic easier to follow. Furthermore, a fail fast break command was inserted to prevent continued searching through the collection after the item required was already found. Further improves execution time of the 'getSummary' method.

**OLD:**

public class StockManager

{

//private Finance stockObj;

private HashMap<Finance, Float> portfolio = new HashMap<Finance, Float>();

FeedParser newParse = new FeedParser();

public void clearPortfolio()

{

portfolio.clear();

}

public boolean addPortfolioEntry(String stockCode, float shareQuantity)

{

Finance stockObj = newParse.parseJSON(stockCode);

if (portfolio.containsKey(stockObj))

{

return false;

}

portfolio.put(stockObj, shareQuantity);

return true;

}

public float getPortfolioTotal()

{

float value = 0;

if (portfolio.isEmpty())

{

return 0;

}

for (Finance stockObj : portfolio.keySet())

{

value += stockObj.getLast() \* portfolio.get(stockObj);

}

return value;

}

public String getSummary()

{

float total = 0.0f;

String summary = "";

**// Create sorted list of stocks...**

**List<String> stockNames = new ArrayList<String>();**

**for (Finance thisStockObj : portfolio.keySet())**

**{**

**stockNames.add(thisStockObj.getName());**

**}**

// Now sort...

Collections.sort(stockNames);

for (String currStockName : stockNames)

{

Finance stockObj = null;

for (Finance thisObj : portfolio.keySet())

{

if (thisObj.getName().equals(currStockName))

{

stockObj = thisObj;

}

}

float thisStockValue = stockObj.getLast();

float subTotal = portfolio.get(stockObj) \* stockObj.getLast();

summary += stockObj.getName() + " :\n";

summary += " " + String.format("%3.0f", portfolio.get(stockObj));

summary += " shares\n";

summary += " Unit Price: £" + String.format("%3.2f", thisStockValue);

summary += "\n";

summary += " Sub Total: £" + String.format("%3.2f",subTotal);

summary += "\n\n";

total += subTotal;

}

summary += "\nTotal Portfolio Value : £" + String.format("%3.2f", total);

return summary;

}

}

**NEW:**

public class StockManager

{

//private Finance stockObj;

private HashMap<Finance, Float> portfolio = new HashMap<Finance, Float>();

**List<String> stockNames = new ArrayList<String>();**

FeedParser newParse = new FeedParser();

public void clearPortfolio()

{

portfolio.clear();

**stockNames.clear();**

}

public boolean addPortfolioEntry(String stockCode, float shareQuantity)

{

Finance stockObj = newParse.parseJSON(stockCode);

if (portfolio.containsKey(stockObj))

{

return false;

}

portfolio.put(stockObj, shareQuantity);

**stockNames.add(stockObj.getName());**

return true;

}

public float getPortfolioTotal()

{

float value = 0;

if (portfolio.isEmpty())

{

return 0;

}

for (Finance stockObj : portfolio.keySet())

{

value += stockObj.getLast() \* portfolio.get(stockObj);

}

return value;

}

public String getSummary()

{

float total = 0.0f;

String summary = "";

// Now sort...

Collections.sort(stockNames);

for (String currStockName : stockNames) // Sorted list of names

{

Finance stockObj = null;

for (Finance thisObj : portfolio.keySet())

{

if (thisObj.getName().equals(currStockName))

{

stockObj = thisObj;

**break; // fail fast**

}

}

float thisStockValue = stockObj.getLast();

float subTotal = portfolio.get(stockObj) \* stockObj.getLast();

summary += stockObj.getName() + " :\n";

summary += " " + String.format("%3.0f", portfolio.get(stockObj));

summary += " shares\n";

summary += " Unit Price: £" + String.format("%3.2f", thisStockValue);

summary += "\n";

summary += " Sub Total: £" + String.format("%3.2f",subTotal);

summary += "\n\n";

total += subTotal;

}

summary += "\nTotal Portfolio Value : £" + String.format("%3.2f", total);

return summary;

}

}

**Refactoring Example 2**

AgileProjectActivity.java class

GitHub Entry : 17/11/2011 12:57am

On commencement of sprint two, it was clear that there were three separate tasks that were to be present in the final solution, and each required it's own activity thread in which to run. This was accomplished by creating by changing the 'AgileProjectActivity' class to be the top-level activity, and abstracting the functionality from this class into a second level activity 'SummaryActivity' instead.

In the refactored code, the 'AgileProjectActivity' is responsible for changing between the required activities depending on the tab selected by the user. This provided a better structure to the code, with separate activity classes representing each of the tabs visible on the final application.