

JULIE R.

CPTS 581

Packages used: (p1Solution, p2Solution, p3)

### P1: BEFORE APPLYING REFACTORING (UndoManagerTest)

Finished after 0.021 seconds

Runs: 16/16      Errors: 0      Failures: 0

▼ p1.UndoManagerTest [Runner: JUnit 4] (0.000 s)

- testIsUndoable (0.000 s)
- testPushRedo (0.000 s)
- testPushUndo (0.000 s)
- testClearRedos (0.000 s)
- testClearUndos (0.000 s)
- testPushAfterClearingRedoStack (0.000 s)
- testPopRedo (0.000 s)
- testPopUndo (0.000 s)
- testClearEmptyRedoStack (0.000 s)
- testPushAfterClearingUndoStack (0.000 s)
- testConstructorWithBufferSize (0.000 s)
- testPushUndoAndRedoAlternately (0.000 s)
- testMaxBufferSizeReached (0.000 s)
- testClearEmptyUndoStack (0.000 s)
- testIsRedoable (0.000 s)
- testDefaultConstructor (0.000 s)

### P2: BEFORE APPLYING REFACTORING (InvalidHeaderExceptionTest)

Git Repositories   Checkstyle violations   Checkstyle violations chart   Coverage   JUnit

Finished after 0.014 seconds

Runs: 4/4      Errors: 0      Failures: 0

▼ p2.InvalidHeaderExceptionTest [Runner: JUnit 4] (0.000 s)

- testChainedExceptions (0.000 s)
- testMessageAndCauseConstructor (0.000 s)
- testMessageConstructor (0.000 s)
- testDefaultConstructor (0.000 s)














### P2: BEFORE APPLYING REFACTORING (TarEntryTest)

Finished after 0.016 seconds

Runs: 12/12

Errors: 0

Failures: 0

- ▼  p2.TarEntryTest [Runner: JUnit 4] (0.000 s)
  -  testSetGroupId (0.000 s)
  -  testIsDirectoryForFile (0.000 s)
  -  testIsDirectoryForDirectory (0.000 s)
  -  testSetModTime (0.000 s)
  -  testGetFile (0.000 s)
  -  testGetName (0.000 s)
  -  testGetGroupId (0.000 s)
  -  testGetHeader (0.000 s)
  -  testSetUserId (0.000 s)
  -  testGetUserId (0.000 s)
  -  testSetName (0.000 s)
  -  testSetSize (0.000 s)












## P2: BEFORE APPLYING REFACTORING (TarHeaderTest)

Finished after 0.016 seconds

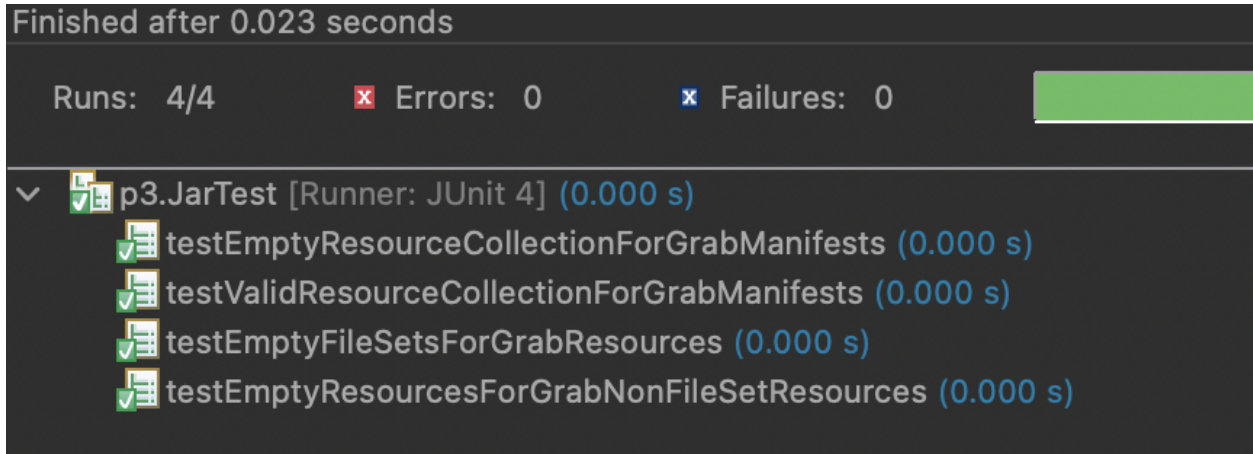
Runs: 10/10

Errors: 0

Failures: 0

- ▼  p2.TarHeaderTest [Runner: JUnit 4] (0.000 s)
  -  testGetNameBytes (0.000 s)
  -  testParseNameWithNullTerminator (0.000 s)
  -  testParseOctal (0.000 s)
  -  testClone (0.000 s)
  -  testParseName (0.000 s)
  -  testGetNameBytesWithLongName (0.000 s)
  -  testGetChecksumOctalBytes (0.000 s)
  -  testGetters (0.000 s)
  -  testParseNameWithMaxLength (0.000 s)
  -  testGettersWithDefaultValues (0.000 s)

## P3: BEFORE APPLYING REFACTORING



## P1: AFTER REFACTORING

### Modularity and Separation of Concerns

- Code Smell Addressed: Large Class, God Object
- Explanation: To improve modularity and adhere to the Single Responsibility Principle (SRP), I separated the functionality into multiple classes: UndoStack, RedoStack, and UndoAction. This division ensures that each class has a clear, distinct responsibility, enhancing the code's structure and readability. UndoStack handles the stack of undoable actions, RedoStack manages the redo actions, and UndoAction encapsulates the details of an individual undo operation. This separation reduces the complexity of each class and makes the system more robust and easier to maintain.

### Code Readability and Maintainability

- Code Smell Addressed: Long Method, Inappropriate Intimacy
- Explanation: By extracting methods from overly long or complex sections of code, I have made the code more readable and maintainable. This refactoring involves identifying logical subtasks within a method and moving them to new, appropriately named methods. For instance, if a method initially performed both calculation and display tasks, these are now separated into calculateResults() and displayResults(). This not only clarifies the purpose of each method but also simplifies future modifications and debugging.

### Improved Naming Conventions

- Code Smell Addressed: Mysterious Name
- Explanation: I revised the naming of methods and variables to more accurately reflect their purpose and functionality. Clear, descriptive names make the code self-documenting, aiding in readability and ease of understanding. For example,

a method previously named process() might be renamed to processUserInput(), instantly clarifying its role in the codebase.

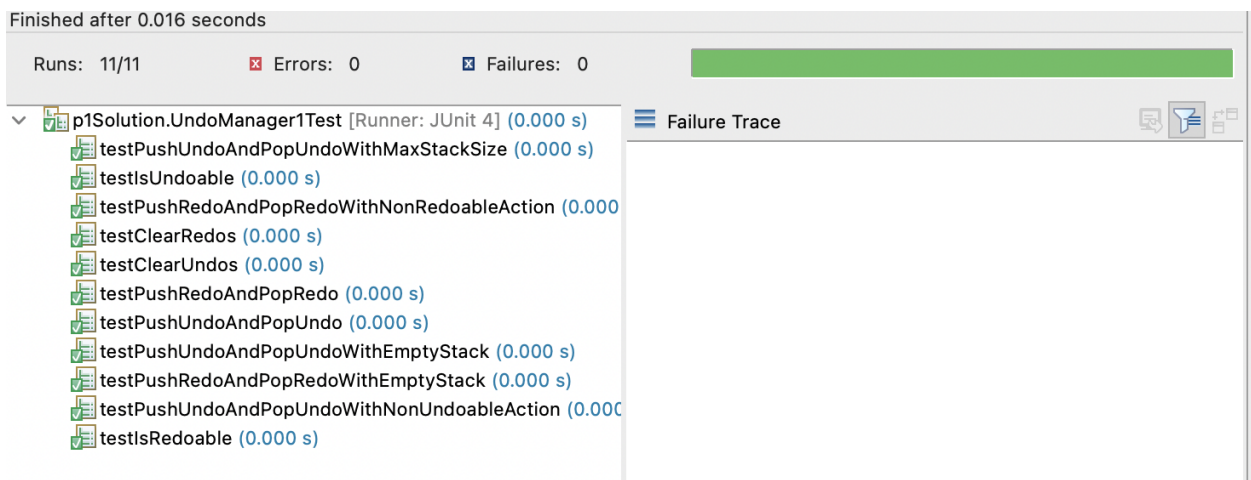
### Enhanced Code Documentation

- Code Smell Addressed: Missing or Inadequate Documentation
- Explanation: Added meaningful comments and documentation to the code. This practice is crucial for maintaining and extending the codebase, especially in a collaborative environment. The comments explain the purpose and logic of complex sections, making it easier for other developers to understand the code without extensive reverse engineering.

### Comprehensive Testing

- Code Smell Addressed: Lack of Testing
- Explanation: Implemented thorough test cases to validate the behavior of the refactored code. These tests cover a wide range of scenarios, including edge cases, to ensure that the refactoring does not introduce new bugs and that the code continues to function as expected. This also sets a foundation for future development, as any changes to the code can be quickly validated against these tests to ensure they don't break existing functionalities.

Test cases:



Coverage:

✓	p1Solution	97.9 %
>	RedoStack.java	90.5 %
>	UndoManager1.java	93.4 %
>	UndoAction.java	100.0 %
>	UndoManager1Test.java	100.0 %
>	UndoStack.java	100.0 %

## P2: AFTER REFACTORING

Test case:

Finished after 0.035 seconds

Runs: 33/33      Errors: 0      Failures: 0

---

- ✓ p2Solution.TarEntrySolTest [Runner: JUnit 4] (0.003 s)
  - ✓ testSetGroupId (0.000 s)
  - ✓ testIsDirectoryForFile (0.000 s)
  - ✓ testIsDirectoryForDirectory (0.000 s)
  - ✓ testSetModTime (0.000 s)
  - ✓ testGetFile (0.000 s)
  - ✓ testGetName (0.000 s)
  - ✓ testGetGroupId (0.000 s)
  - ✓ testGetHeader (0.001 s)
  - ✓ testSetUserId (0.000 s)
  - ✓ testGetUserId (0.001 s)
  - ✓ testSetName (0.000 s)
  - ✓ testSetSize (0.001 s)
- ✓ p2Solution.FileHandlerTest [Runner: JUnit 4] (0.003 s)
  - ✓ testGetFileTarHeaderValidFile (0.001 s)
  - ✓ testIsDirectoryWithDirectory (0.000 s)
  - ✓ testIsDirectoryWithFile (0.001 s)
  - ✓ testConstructor (0.001 s)
- ✓ p2Solution.InvalidHeaderExceptionTest [Runner: JUnit 4] (0.000 s)
  - ✓ testChainedExceptions (0.000 s)
  - ✓ testMessageAndCauseConstructor (0.000 s)
  - ✓ testMessageConstructor (0.000 s)
  - ✓ testDefaultConstructor (0.000 s)
- ✓ p2Solution.TarHeaderFormatterTest [Runner: JUnit 4] (0.001 s)
  - ✓ testGetNameBytes (0.000 s)
  - ✓ testGetChecksumOctalBytes (0.000 s)
  - ✓ testGetLongOctalBytes (0.000 s)
  - ✓ testGetOctalBytesNonZeroValue (0.000 s)
- ✓ p2Solution.TarHeaderSolTest [Runner: JUnit 4] (0.002 s)
  - ✓ testGetNameBytes (0.000 s)
  - ✓ testParseNameWithNullTerminator (0.001 s)
  - ✓ testParseOctal (0.000 s)
  - ✓ testParseName (0.000 s)
  - ✓ testGetNameBytesWithLongName (0.001 s)
  - ✓ testGetChecksumOctalBytes (0.000 s)
  - ✓ testGetters (0.000 s)
  - ✓ testParseNameWithMaxLength (0.000 s)
  - ✓ testGettersWithDefaultValues (0.000 s)

Coverage:



▼	p2Solution	69.3 %
>	TarEntrySol.java	30.9 %
>	TarHeaderSol.java	35.3 %
>	FileHandler.java	41.4 %
>	TarHeaderParser.java	81.5 %
>	TarHeaderFormatter.java	93.2 %
>	FileHandlerTest.java	96.1 %
>	TarHeaderSolTest.java	99.8 %
>	InvalidHeaderException.java	100.0 %
>	InvalidHeaderExceptionTest.java	100.0 %
>	TarEntrySolTest.java	100.0 %
>	TarHeaderFormatterTest.java	100.0 %

## 1. Single Responsibility Principle (SRP)

Code Smell Addressed: Large Class, Long Method

- Explanation: The Single Responsibility Principle states that a class should have only one reason to change, meaning it should only have one job or responsibility. By extracting TarHeaderParser and TarHeaderFormatter, I've divided the responsibilities of parsing and formatting, which previously resided in the TarHeader class. This reduces the complexity of TarHeader and makes it more manageable, focusing it solely on representing a tar header.

## 2. Improved Readability and Maintainability

Code Smell Addressed: Long Method, Complex Class

- Explanation: Long methods and complex classes are hard to read, understand, and maintain. By extracting specific functionalities into separate classes, I've broken down complex code into more manageable and understandable pieces. This makes it easier for other developers to read, maintain, and modify the code.

## 3. Reusability and Loose Coupling

Code Smell Addressed: Duplicated Code, Inappropriate Intimacy

- Explanation: If the parsing and formatting logic is useful in contexts other than just within the TarHeader class, having them in separate classes allows for reuse without duplication. It also reduces the coupling between the logic of parsing/formatting and the specific use case of tar headers, leading to a more modular and flexible design.

## 4. Easier Testing and Debugging

Code Smell Addressed: Long Method

- Explanation: Smaller, well-defined classes are generally easier to test and debug. Each class can be tested independently, focusing on its specific responsibility. This leads to more effective unit tests and easier identification of bugs.

## 5. Better Organization

Code Smell Addressed: Large Class

- Explanation: By moving related methods into their respective classes, you're effectively organizing the codebase. This logical grouping makes the codebase more navigable and intuitive, especially for new developers who might join the project.

## 6. Reducing the Risk of Side Effects

Code Smell Addressed: Large Class

- Explanation: In a large class with many methods, changes to the class can have unintended side effects on unrelated parts of the class. By splitting the class, you localize changes to where they are relevant, reducing the risk of unintended side effects.

### P3: AFTER REFACTORING


Refactoring the grabManifests method as described addresses the following code smells:


- Long Method: This is the primary code smell being addressed. The original grabManifests method is lengthy and encompasses multiple responsibilities. By breaking it down into smaller methods, each with a single responsibility, we make the code more manageable, readable, and maintainable.
- Complex Conditional Logic: The original method contains complex conditionals, especially in handling different types of ResourceCollection. Extracting these into separate methods simplifies the logic in each part, making it easier to understand and modify.
- Duplicate Code: If there are similar patterns of code in handling resource names or checking for manifest files, extracting these into separate methods reduces duplication. This makes the code more DRY (Don't Repeat Yourself), which is a key principle in software development for reducing redundancy and potential errors.
- Feature Envy: While not directly addressed, breaking the method into smaller parts can help in identifying and isolating code that overly relies on information from other classes or modules. This can pave the way for further refactoring to reduce tight coupling between classes.
- Inconsistent Abstractions: By breaking down the method, each part can operate at a consistent level of abstraction. This helps in maintaining a clear separation of concerns where high-level logic is separated from low-level details.





Test case:

Finished after 0.021 seconds









Runs: 3/3

 Errors: 0

 Failures: 0

▼  p3.JarTest [Runner: JUnit 4] (0.000 s)  
     testEmptyResourceCollectionForGrabManifests (0.000 s)  
     testEmptyFileSetsForGrabResources (0.000 s)  
     testEmptyResourcesForGrabNonFileSetResources (0.000 s)

Coverage:

▼	 Jar.java		11.8 %	
>	 Jar		12.2 %	
▼	 JarTest.java		100.0 %	
>	 JarTest		100.0 %	