VU Softwarequalitätssicherung – Beispiel 4

Bericht

4a) FindBugs

Fehler Name	Fundort, Beschreibung
equals() method does not check for null argument	org.junit.experimental.theories.internal.ParameterizedAssertionError.equals(Object obj). Der Parameter der Methode wird nicht auf *NULL*-Wert überprüft. Es kann somit zu einer „*NullPointerException*“ kommen. Lösung durch Überprüfung auf *NULL* zu Beginn.
Incorrect lazy initialization and update of static field junit.runner.BaseTestRunner.fPreferences	junit.runner.BaseTestRunner.getPreferences(). Ein statischer Klassenmember wird, falls der Member noch nicht gesetzt ist, innerhalb der statischen Methoden *getPreferences()* dynamisch erzeugt. Dies kann bei Verwendung von Threads zu Synchronisationsproblemen führen, da die Methode kein Thread-Locking benutzt. Lösung durch Verwendung des Keywords „*synchronized*“. Hierdurch stellt Java sicher, das maximal ein Thread gleichzeitig die Methode betreten darf.
Method junit.framework.Assert.assertEquals(String, int, int) invokes inefficient new Integer(int) constructor; use Integer.valueOf(int) instead	junit.framework.Assert.assertEquals(String message, int expected, int actual). Ein neues Integer-Objekt wird aus einem bestehenden Integer-Wert dynamisch mittels new erzeugt. Diese Vorgehensweise ist jedoch nicht effizient. Bessere wäre die Benutzung der Methode *Integer.valueOf(value),* die durch Caching der JVM das selbe Resultat in kürzerer Zeit liefert. Lösung durch Ersetzen der Konstrukte „*new Integer(value)*“ durch „*Integer.valueOf(value)*“.
org.hamcrest.internal.ArrayIterator.next() can't throw NoSuchElementException	org.hamcrest.internal.ArrayIterator.next(). Die Methode *next()* des Java Interfaces *Iterator* muss die Exception *NoSuchElementException* werfen, sofern der Iterator keine weiteren Elemente mehr enthält. Dies wurde in der Implementierung nicht durchgeführt. Stattdessen wird implizit die Exception *ArrayIndexOutOfBound* geworfen. Lösung durch Abfangen der Exception *ArrayIndexOutOfBound* und werfen einer neuen Exception des Typs *NoSuchElementException*.
junit.textui.TestRunner.runFailed(String) invokes System.exit(...), which shuts down the entire virtual machine	junit.textui.TestRunner.runFailed(String message) FindBugs bemängelt die Verwendung der Methode *System.exit(...)*, die die JVM direkt beendet. Dies ist aber in diesem Fall definitiv so erwünscht, da die Methode nur im Fehlerfall aufgerufen wird und dieser Fehlerfall über den Return-Code des Java Prozesses an das System weiter bzw. zurück kommuniziert werden soll.
int[] passed to varargs method java.util.Arrays.asList(Object[])	FindBugs bemängelt die Erzeugung eines Arrays mit der Größe 1 über die Methode Arrays<T>.asList(T ... a). Dies ist in diesem

	Fall aber so gewollt, wäre aber klarerweise genauso auch ohne Verwendung der Methode möglich.

4b) Refactoring Patterns

Angewandte Refactoring-Patterns	
Pattern Name	Nutzen, Begründung der Auswahl, Name der erstellten Testmethode
Encapsulate Field	Klassenmember *Pizza.Name* wird mittels getting and setting methods realisiert (OO Data Hiding). Methoden: *getName()*, *setName(String name)*
Encapsulate Field	Klassenmember *Pizza.priceInCent* wird mittels getting and setting methods realisiert (OO Data Hiding). Methoden: *getPriceInCent()*, *setPriceInCent(int priceInCent)*
Encapsulate Field	Klassenmember *Pizza.size* wird mittels getting and setting methods realisiert (OO Data Hiding). Methoden: *getSize()*, *setSize(int size)*
Replace Type Code with Subclasses, Replace Conditional with Polymorphism	*Pizza* wird in eine abstrakte Klasse umgewandelt und die drei Klassen *PizzaNormal*, *PizzaChildren*, *PizzaXXL* abgeleitet. Zudem wird die Methode *sizeToString()* ebenfalls abstrakt deklariert und in jeder der Subklassen implementiert. Damit entfällt die switch-Anweisung. Zusätzlich musste noch die Klasse *Runner* und die Tests angepasst werden.
Extract Method, Move Method	Der Klasse *Pizza* wird eine abstrakte Methode *getPoints()* hinzugefügt, die von den jeweiligen Subklassen implementiert werden muss. Diese Methode wird anschließend in *Customer.getOrderSummary()* verwendet.
Extract Method, Move Method	Im Sinne der OO Kapselung wird die abstrakte Methode *Pizza.getPriceMultiplicator()* eingeführt, die den Multiplikator zur Berechnung des Pizzapreises je Pizzagröße definiert. Diese Methode wird entsprechend von allen drei Subklassen implementiert. Entsprechend können die Methoden *Runner.normalToChildrenPrice()* und *Runner.normalToXXLPrice()* entfernt werden.
Extract Method, Move Method	Im Sinne der OO Kapselung wird die Methode *OrderedPizza.calculatePrize()* eingeführt, die den Preis einer *Pizza* berechnet. Neuer Test: *orderPizza_caculatePrize()*
Extract Method, Move Method	Der Klasse *OrderedPizza* wird eine Methode *calculatePoints()* hinzugefügt, die die Pizza Punkte mittels *Pizza.getPoints()* berechnet. Neuer Test: *orderPizza_caculatePoints()*
Replace Temp with Query	Die temporären berechneten Werte in der Methode *Customer.getOrderSummary()* werden in die eigene Methoden *getTotalPrice()* und *getPizzaPoints()* ausgelagert. Neuer Test: *customer_totalPrice()* und *customer_pizzaPoints()*
Encapsulate Field	Klassenmember *Cusomter.orderedPizzas* wird mittels getting methods nach aussen exportiert. Methode: *getOrderedPizzas()*
Extract Class	*Customer.getOrderSummary()* wird in eine eigene Klasse *OrderSummary* ausgelagert. Damit ist diese Klasse ausschließlich (Single Responsibility) für die (visuelle) Ausgabe der „OrderSummary“ zuständig.
Extract Class	Die Validierung der Werte in *OrderWindows.actionPerformed()* wird in eine eigene Klasse *Validator* und dessen Methode *checkValid()* ausgelagert. Diese gibt bei ungültiger Validierung die Fehlermeldung für den Benutzer zurück.
Extract Class	Eine neue Klasse UIHandler wird eingeführt. Diese enthält die Beschriftung und die

	Logik der UI. Durch dieses Refactoring können verschiedene UIs mit dem selben Code verwendet werden. Die Beschriftung der UI-Elemente wird über getting methods nach aussen exportiert. Neue Methoden: *UIHandler .getTxtNewbtn()*, *UIHandler .getTxtCustomername()*, *UIHandler .getTxtAvailablepizzas()*, *UIHandler .getTxtQuantity()*, *UIHandler .getTxtAdd()*, *UIHandler .getTxtOrderoutput()* und *UIHandler .getTxtTitle()*
Extract Method, Move Method	Das Erzeugen der „Todays Pizzas“ wird in die Methode *UIHandler.createTodaysPizzas()* ausgelagert. Dadurch wird redundanter Code bei Verwendung verschiedener UIs vermieden.
Extract Method, Move Method	Die Werte für die UI-Elemente sowie der Customer wird in die Klasse UIHandler über getting und settings Methods ausgelagert. Dadurch enthält die Klasse OrderWindow nun keine Logik des Customers mehr. Neue Methoden: *UIHandler.newCustomer()*, *UIHandler.addOrder(), UIHandler.getName()*, *UIHandler.setName()*, *UIHandler.getSelectedPizza()*, *UIHandler.setSelectedPizza()*, *UIHandler.getQuantity()*, *UIHandler.setQuantity()*, *UIHandler.getOutput()* und *UIHandler.setOutput()*