Adam Shafei Len Williamson Steffen Pegenau

0. Übungszettel in Software Engineering

Aufgabe 3: Erste Berührungspunkte mit der JVM

Listing 1: Methode *isInitValueValid* mit Parameter vom Typ *Integer*

```
1  /**
2   * This method has been defined to allow the sub-classes
3   * of SnmpInt to perform their own control at intialization time.
4   */
5  boolean isInitValueValid(int v) {
6    if ((v < Integer.MIN_VALUE) || (v > Integer.MAX_VALUE)) {
7      return false;
8    }
9    return true;
10 }
```

Listing 2: Methode *isInitValueValid* mit Parameter vom Typ *Long*

```
1  /**
2   * This method has been defined to allow the sub-classes
3   * of SnmpInt to perform their own control at intialization time.
4   */
5  boolean isInitValueValid(long v) {
6    if ((v < Integer.MIN_VALUE) || (v > Integer.MAX_VALUE)) {
7      return false;
8    }
9    return true;
10 }
```

a) Was fällt Ihnen auf?

Beide Methoden unterscheiden sich ausschließlich in der Signatur: eine Variante prüft eine Integer-Variable, die andere eine Long-Variable. Die beiden Methodenrümpfe sind identisch, was gerade bei der Long-Variante überrascht. Erwarten würde man eine Prüfung auf den Long-Wertebereich. Tatsächlich wird aber auch "nur" auf den Integer-Wertebereich geprüft.
Da ein Kommentar fehlt, lässt sich nicht sagen, ob es sich dabei um einen Fehler handelt.
Überhaupt scheint die Prüfung des Wertebereichs bei Integer-Variante überflüssig, da der Compiler *null* oder nicht initialisierte Integer-Werte als Parameter nicht zulässt. Ist eine Integervariable initialisiert, befindet sich der Wert im gültigen Bereich. Berechnungen wie "$2 \cdot Integer.MIN\_VALUE$" liefern zwar kein mathematisch korrektes Ergebnis, aber das Ergebnis ist ein gültiger Integerwert.
Bei der Long-Variante kann die auf den Integer-Wertbereich beschränkte Überprüfung zum Beispiel dann sinnvoll sein, wenn man wissen will, ob man den Wert einer Long-Variable auch als Integer speichern kann.

a) Falls Sie ein Problem identifizieren, was würden Sie machen, um solche Probleme in der Zukunft zu vermeiden?

Mit automatischen Tests könnte man prüfen, ob...

- es richtig ist, dass die Long-Variante nur Werte im Integer-Bereich akzeptiert
- Gibt es Methoden, die über weite Teile identisch sind?

Über Konvetionen könnte man dafür sorgen, dass solche wenig intuitiven Stellen entsprechend kommentiert werden. Um die Einhaltung zu garantieren, könne man den Commit-Prozess um eine Prüfung durch einen anderen Entwickler ergänzen.

c) Was könnte zu dem Problem geführt haben?

Eventuell bestand spontan der Bedarf nach derselben Methode mit einer anderen Signatur, sodass die Methode einfach per Copy & Paste kopiert wurde und dabei vergessen wurde, den Methoderumpf an allen Stellen zu prüfen und anzupassen.
Bei der gegebenenfalls überflüssigen Prüfung des Wertebereichs dachte der Entwickler womöglich einfach zu kompliziert. Oder er wollte/konnte sich nicht auf die Typ-Garantie des Compilers verlassen.

Listing 3: Methode *containsValue*

```
1  /**
2   * Returns true if this attribute set contains the given
3   * attribute.
4   *
5   * @param   attribute  value whose presence in this attribute set is
6   *             to be tested.
7   *
8   * @return  true if this attribute set contains the given
9   *       attribute    value.
10  */
11 public boolean containsValue(Attribute attribute) {
12   return
13     attribute != null &&
14     attribute instanceof Attribute &&
15     attribute.equals(attrMap.get(((Attribute)attribute).getCategory()));
16 }
```

a) Was fällt Ihnen auf?

Da der Compiler beim Aufruf der Methode garantiert, dass der Parameter vom Typ *Attribute* oder einer von diesem Typ erbenden Klasse ist, ist der Aufruf von *instanceof* unnötig. Dasselbe gilt für den Cast in der letzten Überprüfung.

b) Machen Sie ggf. einen Vorschlag für verbesserten Code.

Listing 4: Vorschlag für Methode *containsValue*

```
1  /**
2   * Returns true if this attribute set contains the given
3   * attribute.
4   *
5   * @param   attribute  value whose presence in this attribute set is
6   *             to be tested.
7   *
8   * @return  true if this attribute set contains the given
9   *       attribute    value.
10  */
11 public boolean containsValue(Attribute attribute) {
12   return
13     attribute != null &&
14     attribute.equals(attrMap.get(attribute.getCategory()));
15 }
```