Techniki zabezpieczania kodu - kontrola wyjątków

Temat VIII

Wyjątki w Javie

- To nie są wyjątki
- Rodzaje Throwable
  - błędy
  - wyjątki
    - wyjątki zwykłe
    - wyjątki czasu wykonania

Wyjątki – na co uważać?

- Typowy interfejs Throwable
- printStackTrace() – czy po sprawdzeniu tajnych danych wyrzucamy wyjątki?
- getMessage() – czy opisy są jednorodne ze względu na tajne dane
- to String() – czy typy wyjątków są jednorodne ze względu na tajne dane
- implementuje interfejs `Serializable`

Wyjątki – printStackTrace()

```
java.lang.NullPointerException
at org.tigris.subversion.subclipse.ui.decorator.SVNLightweightDecorator.
   decorateTextLabel(SVNLightweightDecorator.java:362)
at org.tigris.subversion.subclipse.ui.decorator.
   SVNLightweightDecorator.decorate(SVNLightweightDecorator.java:305)
at org.eclipse.ui.internal.decorators.LightweightDecoratorDefinition.
   decorate(LightweightDecoratorDefinition.java:253)
at org.eclipse.ui.internal.decorators.
   LightweightDecoratorManager$LightweightRunnable.
   run(LightweightDecoratorManager.java:71)
at org.eclipse.core.runtime.SafeRunner.run(SafeRunner.java:37)
at org.eclipse.core.runtime.Platform.run(Platform.java:843)
at org.eclipse.ui.internal.decorators.LightweightDecoratorManager.
   decorate(LightweightDecoratorManager.java:336)
at org.eclipse.ui.internal.decorators.LightweightDecoratorManager.
   getDecorations(LightweightDecoratorManager.java:322)
at org.eclipse.ui.internal.decorators.DecorationScheduler$1.
   ensureResultCached(DecorationScheduler.java:338)
at org.eclipse.ui.internal.decorators.DecorationScheduler$1.
   run(DecorationScheduler.java:308)
at org.eclipse.core.internal.jobs.Worker.run(Worker.java:58)
```

Błędy – klasa Error

- Error to podklasa Throwable
- Poważny problem
- Sensowny program nie powinien wyłapywać (uwaga na przekazywane informacje)
- Najważniejsze kategorie:
  - błędy maszyny wirtualnej
  - błędy w linkowaniu dynamicznego kodu
  - błędy w parsowaniu adnotacji
  - błędy w asercjach
  - inne błędy platformy
- Brak wpływu na typ, opis i zapis stosu

Błędy `RuntimeException`

- Sensowny program może wyłapywać
- Nie trzeba wymieniać w klauzulach `throws`
- Przykłady:
  - ClassCastException
  - IndexOutOfBoundsException,
  - ArrayStoreException
  - i inne

JML – Java Modeling Language

- Formalny język specyfikacji dla Javy
- Zapis decyzji implementacyjnych i projektowych
- Sprawdzanie z rzeczywistym kodem
- Cel: JML powinien być łatwy dla programistów

JML – podstawy syntaktyczne

- JML w komentarzach /* . . . */ lub // . . .
- Własności opisywane wyrażeniami boolowskimi
- Drobne rozszerzenia wyrażeń (\old, \forall, \result)

JML – biurokracja

- Specyfikacje w JML-u mogą być tak silne i tak słabe, jak nam się podoba

```
/*@ requires amount >= 0;
    ensures true;
@*/
public int debit(int amount) {
...
}
```

Domyślny post-warunek `ensures true` można opuścić.

Narzędzia dla JML-a

- parsowanie i sprawdzanie typów
- sprawdzanie asercji w czasie wykonania (jmlrac)
- rozszerzone sprawdzanie statyczne, tzn. automatyczna weryfikacja programu (ESC/Java2)
- weryfikacja programów (LOOP, Krakatoa+Why i in.)

Sprawdzanie asercji w czasie wykonania

- Kompilator jmlrac z Iowa State Univ.
  - tłumaczy asercje JML na sprawdzenia w czasie wykonania (wszystkie asercje są sprawdzane i każde ich naruszenie jest zgłaszane jako błąd)
  - tanie i łatwe w adaptacji do istniejącej infrastruktury testowania
  - lepsze testowanie i lepsza odpowiedź (bo więcej własności jest testowane i w większej liczbie miejsc kodu), np. komunikat *Invariant violated in line 8000* po 1 minucie zamiast *NullPointerException in line 2000* po 4 minutach
  - Oczywiście błąd może być w kodzie lub w specyfikacji
  - Narzędzie jmlunit łączy jmlrac i testowanie jednostkowe

Rozszerzone statyczne sprawdzanie

- Narzędzie ESC/Java(2)
- Rozszerzone statyczne sprawdzanie = w pełni zautomatyzowana weryfikacja programów przy pewnych kompromisach, aby zapewnić pełną automatyzację
- Wykonywana jest próba udowodnienia w czasie kompilacji zgodności specyfikacji z kodem

Rozszerzone statyczne sprawdzanie

- Metoda nie jest poprawna – ESC/Java może nie zauważyć istniejącego błędu
- Metoda nie jest pełna – ESC/Java możę ostrzegać o błędach, które nie są możliwe
- ...ale szybko znajduje dużo potencjalnych błędów
- Dobra przy stwierdzaniu braku wyjątków czasu wykonania i weryfikacji dosyć prostych własności

Rodzaje ostrzeżeń w ESC/Java2

- Ostrzeżenia o możliwych wyjątkach czasu wykonania (`Cast`, `Null`, `NegSize`, `IndexTooBig`, `IndexNegative`, `ZeroDiv`, `ArrayStore`)
- To najczęstsze wyjątki czasu wykonania wynikające z problemów w kodowaniu (tzn. nie wyrzucane jawnie)
- To nie wszystkie wyjątki
- Pozostałe w większości jawnie rzucane przez metody biblioteczne

Wyjątki raportowane

- Wyjątki raportowane w Javie (np. `FileNotFoundException`) to wyjątki, które nie są `RuntimeExceptions` lub `Error`
- Wyjątki pojawiające się w ciele metody są wymagane w deklaracjach `throws`
- Narzędzia JML-a sprawdzają, czy deklaracje `throws` są poprawne (jak kompilator)
- Zwykle wyjątki te pojawiają się w klauzulach `signals` w JML-u

Wyjątki raportowane c.d.

- ESC/Java2 sprawdza przez wnioskowanie, czy podany warunek zachodzi
- Domyślna specyfikacja:

  ```
  signals (OccurringException) true;
  ```

- ESC/Java2 zakłada, że wyjątki raportowane niezadeklarowane w `throws` nie będą się pojawiać

Wyjątki nieraportowane

- Wyjątki nieraportowane (np. `NoSuchElementException`) to `RuntimeExceptions`
- Java nie wymaga, aby były deklarowane w klauzulach `throws`
- ESC/Java2 jest bardziej restrykcyjne – pojawi się ostrzeżenie `Exception` jeśli nieraportowany wyjątek zostanie rzucony, ale nie jest zadeklarowany w `throws`
- Ostrzeżenie: obecnie ESC/Java2 zakłada, że niezadeklarowany wyjątek nieraportowany nie będzie wyrzucony, nawet jeśli pojawia się w klauzulu `signals`
- Deklaruj wszystkie nieraportowane wyjątki, jakie mogą być rzucane (zwłaszcza, ponieważ nie można tego sprawdzić)

Ostrzeżenia typu `Cast`

- Ostrzeżenie `Cast` pojawia się, gdy ESC/Java2 nie jest w stanie stwierdzić, że nie będzie wyrzucony wyjątek `ClassCastException`:

```
public class CastWarning {
       public void m(Object o) {
          String s = (String)o;
       }
}
```

daje

```
------------------------------------------------------------------------
CastWarning.java:3: Warning: Possible type cast error (Cast)
                                          String s = (String)o;
------------------------------------------------------------------------
```

Ostrzeżenia typu `Cast` c.d.

Poniżej jest OK:

```
public class CastWarningOK {
        public void m(Object o) {
            if (o instanceof String) {
                String s = (String)o;
            }
        }
}
```

Ostrzeżenia typu `Cast` c.d.

Poniżej też OK:

```
public class CastWarningOK2 {
   //@ requires o instanceof String;
   public void m(Object o) {
      String s = (String)o;
   }
}
```

Ostrzeżenia typu `Null`

- Ostrzeżenie `Null` pojawia się, gdy ESC/Java2 nie jest w stanie stwierdzić, że nie będzie wyrzucony wyjątek `NullPointerException`:

```
public class NullWarning {
    public void m(Object o) {
        int i = o.hashCode();
    }
}
```

daje

```
------------------------------------------------------------
NullWarning.java:3: Warning: Possible null dereference (Null)
                                      int i = o.hashCode();
------------------------------------------------------------
```

Ostrzeżenia typu `Null` c.d.

Poniżej jest OK:

```
public class NullWarningOK {
   public void m(/*@ non_null */ Object o) {
      int i = o.hashCode();
   }
}
```

Ostrzeżenia typu `Null` c.d.

- W wielu warunkach JML-owych mówi się o tym, że referencje nie są `null`. Wprowadza się wygodne skróty:

```
public class Directory {
private /*@ non null @*/ File[] files;
void createSubdir(/*@ non null @*/ String name){
...
Directory /*@ non null @*/ getParent(){
...
```

Ostrzeżenia typu `ArrayStore`

- Ostrzeżenie `ArrayStore` pojawia się, gdy ESC/Java2 nie jest w stanie stwierdzić, że nie będzie wyrzucony wyjątek `ArrayStoreException`:

```
public class ArrayStoreWarning {
   public void m(Object o) {
      Object[] s = new String[10];
      s[0] = o;
   }
}
```

daje

```
------------------------------------------------------------
ArrayStoreWarning.java:4: Warning: Type of right-hand side
possibly not a subtype of array element type (ArrayStore)
                                         s[0] = o;
------------------------------------------------------------
```

Ostrzeżenia typu `ArrayStore` c.d.

Poniżej jest OK:

```
public class ArrayStoreWarningOK {
   public void m(Object o) {
      Object[] s = new String[10];
      if (o instanceof String)
        s[0] = o;
   }
}
```

Pozostałe typy

- `ZeroDiv` – zgłaszany, gdy dzielnik (dzielenie całkowitoliczbowe) może być 0
- `NegSize` – zgłaszany, gdy rozmiar tablicy przy alokacji może być
ujemny
- `IndexNegative` – zgłaszany, gdy indeks tablicy może być ujemny
- `IndexTooBig` – zgłaszany, gdy indeks tablicy może być większy lub równy jej długości

Pozostałe typy c.d.

```
public class Index {
  void m() {
    int i = 0;
    int j = 8/i; // powoduje ostrzeżenie ZeroDiv
    Object[] oo = new Object[i-1];// ostrzeżenie NegSize
    oo = new Object[10];
    i = oo[-1].hashCode(); // ostrzeżenie IndexNegative
    i = oo[20].hashCode(); // ostrzeżenie IndexTooBig
  }
}
```

Ostrzeżenie o wyjątkach

```
public class Ex {
   public void m(Object o) {
      if (!(o instanceof String))
        throw new ClassCastException();
   }
}
```

daje

```
------------------------------------------------------------
Ex.java:4: Warning: Possible unexpected exception (Exception)
Execution trace information:
   Executed then branch in "Ex.java", line 3, col 32.
   Executed throw in "Ex.java", line 3, col 32.
------------------------------------------------------------
```

Ostrzeżenie o wyjątkach c.d.

Można to wyłączyć:

- deklarując wyjątek w klauzuli `throws` lub
- przez
  `//@ nowarn Exception;`
  w wierszu, który wywołuje wyjątek lub
- przy pomocy opcji wywołania `-nowarn Exception`

Wyjątki dopuszczane przez specyfikacje

- Domyślnie metoda może rzucać wyjątki, ale tylko te z klauzuli `throws`, zatem

  ```
  //@ requires 0 <= amount && amount <= balance;
  public int debit(int amount)
              throws BankException { ... }
  ```

  ma implicite klauzulę:

  ```
  signals (BankException) true;
  ```

  oraz klauzulę:

  ```
  signals (Exception e) e instanceof BankException;
  ```

Wyjątki dopuszczane przez specyfikacje c.d.

- Domyślnie metoda może rzucać wyjątki, ale tylko te z klauzuli `throws`, zatem

  ```
  //@ requires 0 <= amount && amount <= balance;
  public int debit(int amount) {
  ...
  }
  ```

  ma domyślnie klauzulę

  ```
  signals (Exception) false;
  ```

  Przy okazji – `debit` nie może wyrzucić także nieraportowanego wyjątku, choć Java nie wymaga dla takich wyjątków raportu w `throws`

Pozbywanie się wyjątków

- W celu pozbycia się konkretnego wyjątku `SomeException`:
  1. usuń go z klauzuli `throws` (możliwe tylko dla nieraportowanych wyjątków)
  2. dodaj jawne

     ```
     signals (SomeException) false;
     ```
  3. ogranicz zbiór dozwolonych wyjątków używając postwarunku takiego jak:

     ```
     signals (Exception e) e instanceof E1 ||
                             ... ||
                           e instanceof En;
     ```

     lub równoważnie skróconej wersji powyższego

     ```
     signals_only E1,..., En;
     ```

Pozbywanie się wyjątków

- Aby pozbyć się wszystkich wyjątków
  1. usuń wszystkie wyjątki z klauzuli `throws` (możliwe tylko dla wyjątków nieraportowanych)
  2. dodaj jawne

     ```
     signals (Exception) false;
     ```
  3. użyj słowa kluczowego `normal_behavior`, aby pozbyć się wszystkich wyjątków

     ```
     /*@ normal_behavior
         requires ...
         ensures ...
     @*/
     ```

     `normal_behavior` ma implicite

     `signals (Exception) false`

„Może” a „musi” rzucić wyjątek

- Uwaga na różnicę między
  1. jeśli zachodzi P, to wyrzucone ma być `SomeException`
  2. jeśli `SomeException` jest wyrzucone, to zachodzi P
- Łatwo te rzeczy pomylić
- Wyrażanie
  1. za pomocą `exceptional_behavior`
  2. za pomocą `signals`

„Może” a „musi” rzucić wyjątek c.d.

- Przykład użycia

```
/*@ exceptional_behavior
        requires amount > balance;
        signals (BankException e)
                 e.getReason.equals("Amount too big")
@*/
public int debit(int amount) {
...
}
```

mówi, że `BankException` musi być wyrzucone, gdy `amount > balance`.

„Może” a „musi” rzucić wyjątek c.d.

- Klauzula `normal_behavior` ma domyślnie

  `signals(Exception) false;`

  klauzula `exceptional_behavior` ma domyślnie

  `ensures false`

„Może” a „musi” rzucić wyjątek c.d.

- To samo, trochę inaczej

```
/*@ requires true;
    ensures \old(amount<=balance) && ...
    signals (BankException e)
                   \old(amount>balance) && ...
@*/
public int debit(int amount) throws BankException {
...
}
```

Wyjątki – morał

- Morał: dla uproszczenia zabronić wyjątków w specyfikacjach, gdziekolwiek to możliwe, np. dla

```
public void arraycopy(int[] src, int destOffset,
                      int[] dest, int destOffset,
                      int lenght)
               throws NullPointerException,
                      ArrayIndexOutOfBoundsException
```

  napisać specyfikację, która zabrania wyrzucania wyjątków, a podawać `exceptional_behaviour` jeśli to jest gdzieś naprawdę potrzebne