Programowanie i projektowanie obiektowe

Klasy zagnieżdżone, interfejsy
Jak programować poprawnie?

Paweł Daniluk

Wydział Fizyki

Jesień 2012

Zasięg widoczności

Zmienne i klasy są widoczne tylko w obrębie zasięgu, w którym zostały zadeklarowane.
W Javie zasięgi obliczane są statycznie.

Zasięgi (w kolosalnym uproszczeniu) wyznaczane są nawiasami klamrowymi.

```
class Foo {
    int i;
    void bar() {
        int val=100;
        while(val>2) {
            {
                double val;
                ...
            }
            val--;
        }
     }
}
```

Zasięg widoczności – statycznie czy ~~dynamicznie~~

Widoczność zmiennych może zależeć od miejsca definicji zasięgu (podejście statyczne) lub miejsca jego wywołania (podejście dynamiczne).

To nie jest Java!

```
int x = 0;
int f() { return x; }
int g() { int x = 1; return f(); }
```

W Javie zasięgi obliczane są statycznie.

Modyfikator `static`

`static` oznacza, że element klasy nie należy do żadnej jej instancji.

```
class Foo {
    static int bar;
    float baz;
}

Foo f=new Foo();

//Poprawnie
Foo.bar=10
f.baz=Foo.bar;

//Niepoprawnie
Foo.baz;
f.bar;
```

Modyfikator `static` c.d.

Statyczne metody nie mają dostępu do atrybutów instancji.

```
class Foo {
    static int bar;
    float baz;

    void qux() { //Dobrze
        int i=bar;
        baz=2*baz;
    }

    static void bag() {
        baz+=3.14;    // Źle
        int j=2*bar; // Dobrze
    }
}
```

Klasy zagnieżdżone

Zwykłe klasy (niezagnieżdżone)

```
class Foo {
// Class members
}
```

Klasy zagnieżdżone

```
class Foo { // Top-level class
    class Bar { // Nested class
    }
}
```

Klasy zagnieżdżone c.d.

Klasy lokalne

```
class Foo {
    void bar() {
        class Foobar {  // Local class within a method
        }
    }
}
```

Klasy anonimowe

```
class Foo {
    void bar() {
        new Object() {
            // Creation of a new anonymous class extending Obje
        };
    }
}
```

Interfejsy

Interfejs to szczególny przypadek klasy abstrakcyjnej, która nie implementuje żadnych metod.

Przykład

```
class Pracownik extends Osoba
        // dziedziczenie po klasie
class Samochód implements Pojazd, Towar
        // dziedziczenie po kilku interfesjach
class Chomik extends Ssak implements Puchate, DoGłaskania
        // dziedziczenie po klasie i kilku interfejsach
```

Interfejsy c.d.

Przykład

```
interface ActionListener {
      void actionSelected(int action);
}

interface RequestListener {
    int requestReceived();
}

class ActionHandler implements ActionListener, RequestListener {
    void actionSelected(int action) {}

    public int requestReceived() {}
}

//Calling method defined by interface
RequestListener listener = new ActionHandler(); /*ActionHandler can be
                                    represented as RequestListener...*/
listener.requestReceived(); /*...and thus is known to implement
                            requestReceived() method*/
```

Praktycznie nie zdarza się, żeby program zadziałał poprawnie przy pierwszym uruchomieniu.

Praktycznie nie zdarza się, żeby program zadziałał poprawnie przy pierwszym uruchomieniu.

Jak pisać możliwie poprawne programy?
Jak szukać błędów?

Jak pisać możliwie poprawne programy?

Metoda małych kroków

1. Zacznij od małego programu, który robi coś widocznego.
2. Dodawaj po kilka linii kodu i za każdym razem sprawdzaj, jak działa.
3. Rób tak, aż skończysz.

Każda zmiana jest przetestowana i poprawna.
Łatwo lokalizować błędy.

Strategie

Opakowywanie i uogólnianie

Jeżeli widzisz kawałek kodu, który robi coś spójnego i (być może) powtarzalnego, zrób z niego metodę.

Szybkie prototypowanie

Na początek możesz stosować rozwiązanie uproszczone np. mniej wydajne.

Bottom-up

Zacznij od małych klocków.

Top-down

Zacznij od rozplanowania całości, następnie wypełniaj kawałki treścią.

Warto wyposażać każdą klasę w metodę `toString`.

Częste błędy

Zbyt duże przyrosty

Duże kawałki nowego kodu są trudne do poprawiania.

Upór

Nie należy przywiązywać się do błędnego kodu.

Błądzenie losowe

Wprowadzanie losowych zmian donikąd nie prowadzi.

Uleganie komunikatom kompilatora

Komunikaty o błędach mogą wprowadzać w błąd.

Błędy składniowe

Kompilator zwraca setki błędów

Zazwyczaj tylko pierwszy jest istotny.

Dziwny komunikat o błędzie

- Sprawdź nawiasy i klamry. Metody muszą być wewnątrz klas, a instrukcje wewnątrz metod.
- Wielkość liter jest istotna.
- Średniki.
- Cudzysłowy (wokół łańcuchów znakowych) i apostrofy (wokół pojedynczych znaków).
- Zgodność typów w przypisaniach. Po lewej stronie musi być l-wartość.
- Liczba, typy i kolejność argumentów w wywołaniach metod.
- Czy wartość zwrócona przez metodę jest wykorzystana?
- Czy typ obiektu odpowiada wywoływanej metodzie? Czy metoda statyczna jest poprzedzona nazwą klasy?
- Czy nie ma odwołań do elementów instancji w metodzie statycznej?

Błędy składniowe c.d.

Metoda ostatniej szansy

Wyszukiwanie binarne.

Nie zawsze warto słuchać kompilatora

"Class Golfer must be declared abstract. It does not define int compareTo(java.lang.Object) from interface java.lang.Comparable."

Błędy wykonania

Śledzenie przebiegu wykonania

`System.out.println` doskonale się do tego nadaje.

Program się zawiesza

Nieskończona pętla lub rekurencja.
Należy zlokalizować problematyczną pętlę i sprawdzić co się dzieje z warunkiem.

```
while (x > 0 && y < 0) {
    // do something to x
    // do something to y

    System.out.println("x: " + x);
    System.out.println("y: " + y);
    System.out.println("condition: " + (x > 0 && y < 0));
}
```

Błędy wykonania – wyjątki

`NullPointerException`

Próba dostępu do atrybutu lub metody referencji równej `null`. Być może zmienna jest niezanicjalizowana.

`ArrayIndexOutOfBoundsException`

Wyjście poza zakres tablicy. Czy tablica nie jest za mała?

`StackOverFlowException`

Nieskończona rekurencja.

`ArithmeticException`

Najczęściej dzielenie przez zero.

Błędy wykonania – kiedy wszystko zawodzi

Zmniejsz rozmiar danych.

Usuń nieistotne fragmenty kodu. Znajdź minimalną wersję, która zawiera błąd.

Jeżeli zmiana, która nie powinna mieć efektu, zmienia działanie programu, należy zwrócić na to uwagę.

Błędy logiczne – program działa, ale źle

Należy zrozumieć, co program naprawdę robi.

- Czy program ewidentnie powinien coś robić, ale nie robi?
- Czy robi coś czego nie powinien?
- Czy jakiś kawałek kodu daje niespodziewany rezultat?

Częste błędy

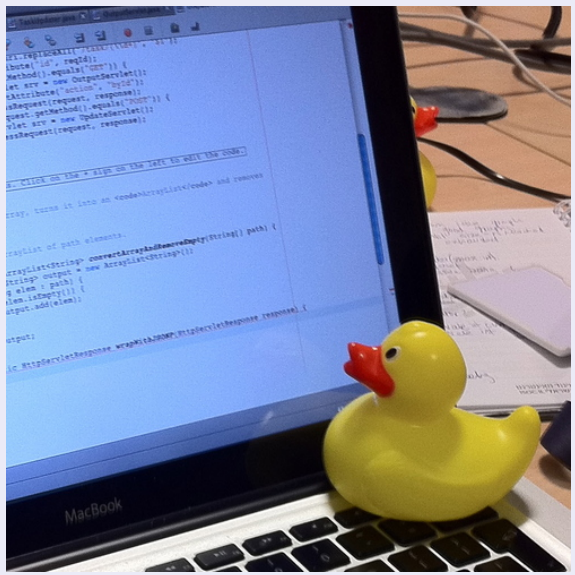
- Dzielenie liczb całkowitych daje całkowity wynik.
- Operacje na liczbach zmiennoprzecinkowych nie są dokładne.
- Operator przypisania (=) zamiast (==).
- Dla typów obiektowych == oznacza tożsamość.
- Dziedziczenie może prowadzić do wywoływania innych metod niż się wydaje.

Kiedy wszystko zawodzi

Komputery emitują fale zaburzające pracę mózgu, które powodują:

- frustrację i gniew
- przesądność ("komputer mnie nienawidzi") i myślenie magiczne ("program działa wyłącznie, gdy trzymam się za lewe ucho")
- pesymizm ("ten program jest beznadziejny")

Metoda gumowej kaczki



Zadanie 1 – Testy

Zadanie

Tworząc odpowiednie przykłady przetestuj działanie mechanizmów przedstawionych na wykładzie.

Wskazówka

Konstrukcje niepoprawne (np. próby dostępu do atrybutów prywatnych) po przetestowaniu zasłoń komentarzem.