Materiał pomocniczy do kursu „Podstawy programowania”

Autor: Grzegorz Góralski
ggoralski.com

Klasy i obiekty cz II

Hermetyzacja, mutatory, akcesory, ArrayList

Rozwijamy aplikację

- Chcemy, aby obiekty klasy `Gatunek`, mogły przechowywać informację na temat chromosomów wchodzących w skład kariotypu.
- Dodamy pole, które będzie przechowywać tablicę obiektów typu `Chromosom`
- Obiekt klasy `Chromosom`, będzie:
  - przechowywał informacje:
    - nr chromosomu
    - długość 1 ramienia
    - długość 2 ramienia
    - informację, czy chromosom płciowy
    - oznaczenie (X/Y/B etc.)
    - uwagi
  - udostępniał metody:
    - podająca sumaryczną długość chromosomu
  - posiadał konstruktory:
    - bez argumentów
    - pobierający numer i długość obu ramion (osobno),
    - pobierający wszystkie dane, oprócz uwag

Klasa Chromosom

```
package gatunki;
public class Chromosom {
	int numer;
	double ramie1;
	double ramie2;
	boolean plciowy;
	String typ;
	String uwagi;
	public Chromosom() {
		this.numer =0;
		this.ramie1 = 0.0;
		this.ramie2 = 0.0;
		this.plciowy = false;
		this.typ = null;
		this.uwagi = null;
	}
	public Chromosom(int numer, double ramie1, double ramie2) {
		this.numer = numer;
		this.ramie1 = ramie1;
		this.ramie2 = ramie2;
		this.plciowy = false;
		this.typ = null;
		this.uwagi = null;
	}
	public Chromosom(int numer, double ramie1, double ramie2, boolean plciowy, String typ) {
		this.numer = numer;
		this.ramie1 = ramie1;
		this.ramie2 = ramie2;
		this.plciowy = plciowy;
		this.typ = typ;
		this.uwagi = null;
	}
	public double getCalaDlugosc() {
		return ramie1+ramie2;
	}
}
```

cd..

- Do klasy **Gatunek** dodajemy:

```
Chromosom[] kariotyp;
......
public void drukujKariotyp() {
      for (int i=0; i<kariotyp.length; i++) {
          Chromosom chromosom = kariotyp[i];
          System.out.println("Chromosom "+chromosom.numer+
                  " ramie 1: "+chromosom.ramie1+
                  " ramie 2: "+chromosom.ramie2);
      }
  }
```

- Do klasy **Gatunki** dodajemy:

```
Chromosom[] kariotyp = {new Chromosom(1, 3.0, 4.0),
                        new Chromosom(2, 4.0, 4.2),
                        new Chromosom(3, 5.0, 7.5)};
gatunek3.kariotyp = kariotyp;
gatunek3.drukujKariotyp();
```

Hermetyzacja: mutatory i akcesory

- Dotychczas zmienialiśmy wartości pól i pobieraliśmy je bezpośrednio, taka możliwość nie zawsze jest pożądana, np:
  - niektóre dane nie powinny być dostępne bezpośrednio (np. z powodów bezpieczeństwa, albo dlatego, że mają one wyłącznie wewnętrzne znaczenie)
  - przed przekazaniem wartości zmiennej, warto sprawdzić czy jest ona prawidłowa/sensowna
  - przed wprowadzeniem danych może być konieczne ich sprawdzenie (np. liczba chromosomów nie powinna być < 0)
  - po wprowadzeniu wartości danej chcemy przeprowadzić jakieś operacje, np. zmienić inne dane
  - .....

Hermetyzacja: mutatory i akcesory

- Rozwiązanie:
  1. Sprawić, by pola nie były widoczne z zewnątrz (bezpośrednio) - jest to tzw. **hermetyzacja**
  2. Stworzyć odpowiednie metody które będą odpowiedzialne za:
     - wprowadzanie i zmianę danych - tzw. **mutatory** (***set**ters*)
     - pobieranie danych - tzw. **akcesory** (***get**ters*)

`private` oznacza, że zmienna jest prywatna, czyli nie jest dostępna bezpośrednio z zewnątrz

```
private int numer;

public int getNumer() {
   return numer;
}
public void setNumer(int numer) {
   this.numer = numer;
}
```

akcesor, wg. konwencji zaczyna się od `get`

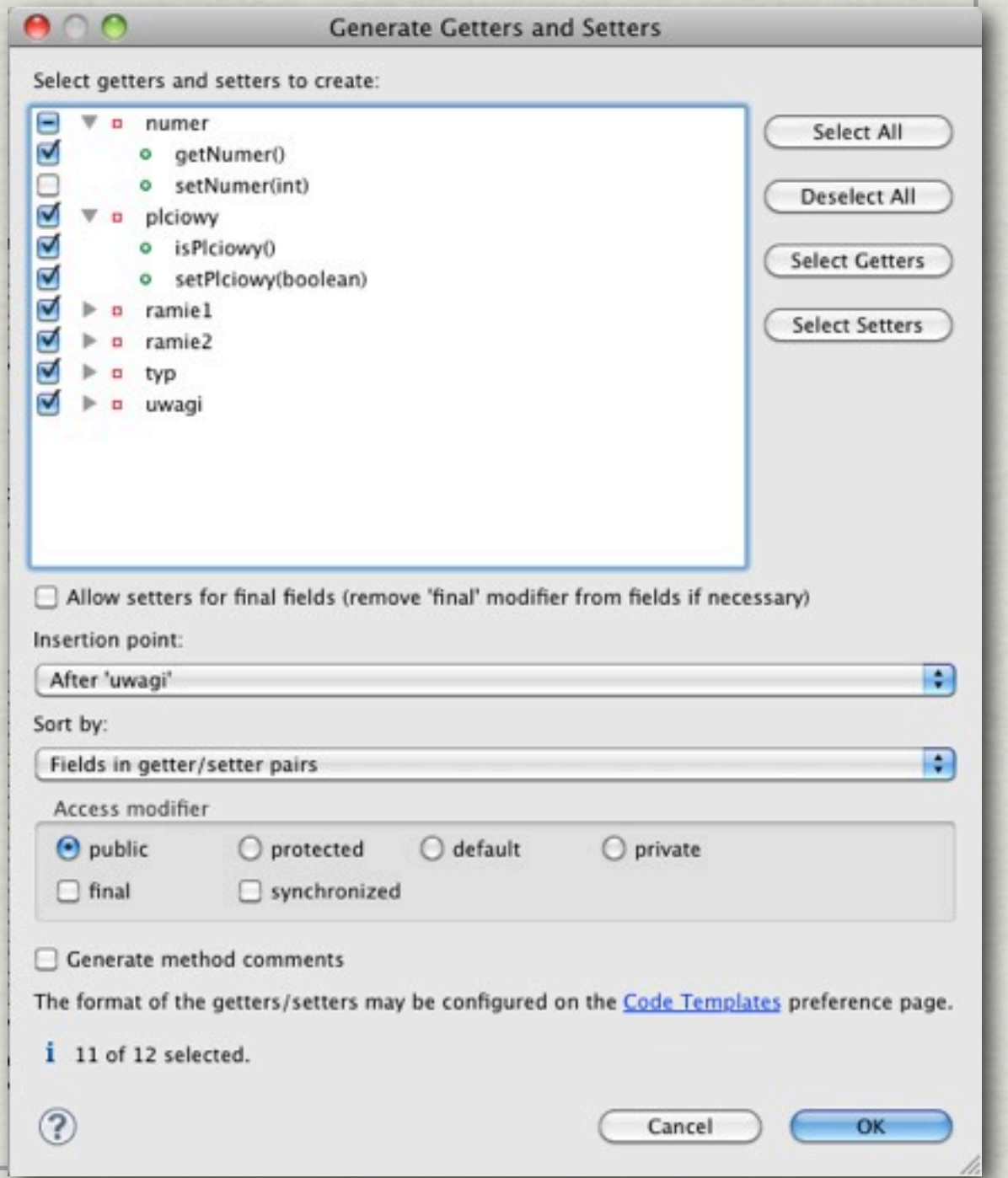
mutator, wg. konwencji zaczyna się od `set`

Hermetyzacja: mutatory i akcesory

- Uwagi:
  - akcesor dla pól typu `boolean` zaczyna się od `is` np. `isPlciowy`
  - pola nie muszą mieć **zarówno** akcesorów i mutatorów, czasem wystarczy jedna z metod, albo żadna (gdy zmiany i dostęp z zewnątrz do zmiennej jest niepożądany)
  - podobnie można zamykać dostęp dla metod, zmieniając `public` na `private`

Hermetyzacja: mutatory i akcesory

- Eclipse ułatwia nam tworzenie mutatorów i akcesorów
- Z menu należy wybrać:
  Source ->
  Generate Getters and Setters
- Wybieramy które z metod chcemy wygenerować, gdzie je umieścić, etc...
- Eclipse utworzy je automatycznie



program cd...

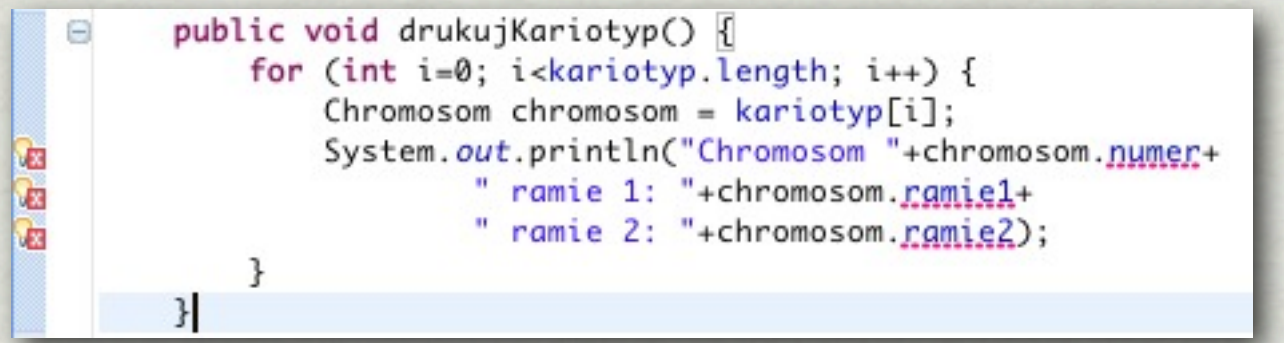
- Zmodyfikujmy teraz klasę `Chromosom`:
  - zmieńmy wszystkie pola na prywatne,
  - wygenerujmy dla nich mutatory i akcesory
  - zapiszmy plik

```
private int numer;
    private double ramie1;
    private double ramie2;
    private boolean plciowy;
    private String typ;
    private String uwagi;

    public int getNumer() {
        return numer;
    }
    public void setNumer(int numer) {
        this.numer = numer;
    }
    public double getRamie1() {
        return ramie1;
    }
....
```

program cd...

- W klasie `Gatunek` Eclipse sygnalizuje błędy:



- Poprawiamy kod:

```
public void drukujKariotyp() {
    for (int i=0; i<kariotyp.length; i++) {
        Chromosom chromosom = kariotyp[i];
        System.out.println("Chromosom "+chromosom.getNumer()+
                " ramie 1: "+chromosom.getRamie1()+
                " ramie 2: "+chromosom.getRamie2());
    }
}
```

program cd...

- Skoro mamy już mutatory i akcesory, zróbmy z tego jakiś pożytek
- W klasie `Chromosom` poprawmy mutator dla pola `numer` (chromosomu), tak, aby nie mógłby być mniejszy od 1:

```
public void setNumer(int numer) {
    if (numer>=1) this.numer = numer;
    else System.out.println("Numer chromosomu nie może być mniejszy od 1!");
}
```

- Podobnie zróbmy z mutatorami dla długości ramion:

```
public void setRamie1(double ramie1) {
    if (ramie1<0) this.ramie1 = ramie1;
    else System.out.println("Długość ramienia nie może być mniejsza od 0!");
}

public void setRamie2(double ramie2) {
    if (ramie2<0) this.ramie2 = ramie2;
    else System.out.println("Długość ramienia nie może być mniejsza od 0!");
}
```

Modyfikatory (specyfikatory) dostępu

- Mogą odnosić się do klas, pól i metod:
  - `public` (publiczne) - swobodnie dostępne
  - `private` (prywatne) - dostępne z wnętrza klasy
  - `protected` (chronione) - dostępne z wnętrza klasy, klas potomnych, pakietu,
  - `domyślne` (pakietowe) - dostęp w obrębie klasy i pakietu

Dodawanie pakietów

- Bardzo często w programie chcemy użyć klas (publicznych) znajdujących się w innych pakietach.
- Można to zrobić na dwa sposoby:
  - `java.util.ArrayList lista = new java.util.ArrayList();`
  - `import java.util.ArrayList;`

    `...`

    `ArrayList lista = new ArrayList();`
- Jeśli chcemy zaimportować wszystkie klasy z pakietu:
  - `import java.util.*;`

Listy tablicowe: `ArrayList`

- Tablice nie pozwalają na zmianę ich rozmiaru, a często jest to potrzebne.
- Rozwiązaniem jest lista tablicowa: `ArrayList`

```
import java.util.ArrayList;
...
ArrayList lista = new ArrayList();
```

- Można od razu określić rodzaj przechowywanych obiektów:

```
ArrayList<String> lista = new ArrayList<String>();
```

Niektóre metody klasy `ArrayList`

- Wydruk zawartości:
```
System.out.println(lista);
```
- Dodawanie zawartości:
```
lista.add("a");
lista.add("b");
```
- Dodawanie zawartości w polu o określonym indeksie:
```
lista.add(1, "c");
```
- Czyszczenie zawartości:
```
lista.clear();
```
- Pobranie elementu znajdującego się pod danym indeksem:
```
lista.get(2);
```
- Usunięcie elementu znajdującego się pod danym indeksem:
```
lista.remove(2);
```
- Zmiana elementu znajdującego się pod danym indeksem:
```
lista.set(0, "u");
```
- Pobranie wielkości:
```
lista.size();
```

Zadanie

- Poprawić i udoskonalić program Gatunki, sugestie:
  - zastosować hermetyzację w pozostałych klasach
  - zmienić tablice na listy tablicowe (ArrayList)
  - zaimplementować manipulacje na liście chromosomów i modyfikację chromosomów.