13. Dziedziczenie

1 Dziedziczenie

Dziedziczenie (ang. *inheritance*) jest mechanizmem umożliwiającym współdzielenie funkcjonalności między klasami. Klasa pochodna (ang. *subclass, derived class, child class*) dziedziczy po klasie bazowej (ang. *base class, superclass*) jej atrybuty oraz metody.

Zadanie 01 Stwórz nowy projekt i dodaj do niego dołączone pliki `Game.java`, `Move.java` i `Paper.java`. Jeżeli korzystasz z IDE konieczne może być włączenie klas do pakietu.

Zadanie 02 Dopisz publiczne klasy `Rock` oraz `Scissors` dziedziczące po `Move`, analogicznie jak klasa `Paper`. Czy możesz utworzyć instancje klas `Paper`, `Rock`, `Scissors`? A co z klasą `Move`?

2 Klasy abstrakcyjne

W języku Java *klasa abstrakcyjna* to klasa opatrzona modyfikatorem `abstract`. Klasa taka może, ale nie musi posiadać *metod abstrakcyjnych*. Nie można tworzyć obiektów takiej klasy, a jedynie po niej dziedziczyć.

Jeżeli klasa posiada przynajmniej jedną metodę abstrakcyjną (odpowiednik metody czysto wirtualnej z języka C++) bez zdefiniowanego ciała metody, to musi być również klasą abstrakcyjną.

Zadanie 03 Przerób klasę `Move` na klasę abstrakcyjną, w szczególności metoda `beats` powinna być abstrakcyjna. Czy teraz da się utworzyć obiekty odpowiednich klas?

Zadanie 04 Dopisz statyczną metodę `getRandom()` do klasy `Move`, która z prawdopodobieństwem $\frac{1}{3}$ zwróci obiekt klasy `Paper`, `Rock` lub `Scissors`. Wykorzystaj klasę `Random`.

Odkomentuj odpowiednie linie w pliku `Game.java` dla sprawdzenia.

Zadanie 05 W klasach pochodnych klasy `Move` przesłoń metodę abstrakcyjną `beats` tak, aby implementowała logikę gry *Papier, kamień, nożyce*. Na przykład, jeżeli w powyższej metodzie dla klasy `Paper` argument `move` będzie instancją obiektu `Rock`, zwracana będzie prawda. Wykorzystaj operator `instanceof`.

Odkomentuj odpowiednie linie w pliku `Game.java`, aby sprawdzić działanie zaimplementowanych funkcji.

Operator `instanceof` pozwala sprawdzić typ obiektu:

```
1 Foo bar = new Foo();
2 if (bar instanceof Foo) {
3     // obiekt bar jest instancja klasy Foo
4 }
```

3 Klasy wewnętrzne

Zadanie 06 Do pliku `Game.java` dodaj prostą klasę `Player` z dwoma atrybutami określającymi nazwę gracza oraz liczbę jego punktów. Zamień odpowiednie linie w funkcji `main()` tak, aby wykorzystać nowoutworzoną klasę.

Umieść klasę `Player` wewnątrz klasy `Game` lub wewnątrz funkcji `main`. Czy kod się skompiluje?

Zadanie 07 domowe (1 pkt) Rozszerz grę *Papier, kamień, nożyce* do wersji *Papier, kamień, nożyce, jaszczurka, Spok*. Zasady gry opisane są na Wikipediihttp://pl.wikipedia.org/wiki/Papier,_kamie%C5%84,_no%C5%BCyce,_jaszczurka,_Spock oraz demonstruje ją video: http://www.youtube.com/watch?v=fqlDc2VICZ0.

Dodaj możliwość gry dla jednego gracza, w której komputer losuje figurę z prawdopodobieństwem $\frac{1}{5}$.

Po rozgrywce wypisz statystyki informujące ile i jakich figur użyto. Zaimplementuj tę funkcjonalność wykorzystując elementy programowania obiektowego, np. poprzez zdefiniowanie statycznej metody w klasie `Move` oraz utworzenie tablicy klas lub obiektów różnego typu.

4 Multimetody*

Multimetody (ang. *multimethods, multiple dispatch*) to mechanizm rozszerzający klasyczny polimorfizm poprzez oparcie wyboru odpowiedniej implementacji metody na dynamicznym, a nie statycznym, typie argumentów. Metody takie intuicyjnie można rozumieć jako metody wirtualne należące do kilku klas.

Przeanalizuj plik `MultipleDispatch.java`.

Zadanie 08 dodatkowe (1 pkt) Zmodyfikuj kod klasy `Move` oraz klas pochodnych z gry *Papier, kamień, nożyce* tak, aby wewnątrz metody `beats` nie używać operatora `instanceof`. Wykorzystaj mechanizm multimetod.

Jak inaczej, bez wykorzystania multimetod, a w dalszym ciągu wykorzystując mechanizmy programowania obiektowego, można zasymulować funkcjonalność operatora `instanceof`?