10 lutego 2012

Programowanie obiektowe

Lista 1.

Zaprogramuj dwa z poniższych zadań w języku w C, Pascalu czy w Pythonie, jednak **bez** użycia elementów obiektowych C++ czy Delphi.

Za każde zadanie na tej liście można otrzymać do 2 punktów.

Zadanie 1. Zadeklaruj typ `Kolekcja`, który może implementować albo *zbiór* przechowując elementy bez powtórzeń, albo *torbę*, przechowując elementy z powtórzeniami. Przyjmujemy, że przechowywane elementy są zadeklarowane deklaracją `typedef Elem ...` (`Elem` może być dowolnym typem). Zaimplementuj procedurę `void wstaw(Kolekcja **k, Elem e)` oraz funkcję `int szukaj(Kolekcja *k, Elem e)` zwracającą liczbę znalezionych elementów. Przyjmujemy, że argumenty `e` są zmiennymi automatycznymi. Zmienna typu `Kolekcja` powinna być inicjowana za pomocą dwóch procedur: `zbiór(Kolekcja **k)` albo `torba(Kolekcja **k)`. Sposób inicjacji determinuje działanie funkcji `wstaw` i `szukaj`.

Zadanie 2. Zadeklaruj typ `typedef Figura ...`, który może reprezentować figury geometryczne: punkt, okrąg lub kwadrat, wraz z ich położeniem w dwuwymiarowym układzie współrzędnych. Przyjmij, że pole `typfig` typu wyliczeniowego wyznacza rodzaj reprezentowanej figury geometrycznej. Zdefiniuj trzy procedury: `narysuj(Figura *f)` (wystarczy, że wypisze odpowiedni komunikat w trybie tekstowym), `przesuń(Figura *f, float x, float y)` (przesuwa figurę o zadany wektor) i `int zawiera(Figura *f, float x, float y)` sprawdzającą, czy figura zawiera punkt o zadanych współrzędnych. Podczas oceny programu wskaż miejsca, które wymagają modyfikacji, jeśli rozszerzymy typ `Figura` o możliwość reprezentacji trójkątów. Zadeklaruj również odpowiednie procedury inicjujące zmienne typu `Figura`.

Zadanie 3. Zaprojektuj własny typ `Zespolone`, tj. podaj odpowiednie struktury danych służące do przechowywania wartości tego typu, oraz nagłówki funkcji odpowiadających czterem standardowym operacjom arytmetycznym na liczbach zespolonych. Zaimplementuj dwie możliwe realizacje takich funkcji: w pierwszej funkcja zwraca wskaźnik do nowoutworzonego elementu tego typu; w drugiej funkcja modyfikuje jeden z argumentów.

Zadanie 4. Zdefiniuj własny typ `DrzewoBinarne` reprezentujące drzewo binarnych poszukiwań. `DrzewoBinarne` przechowujące w węzłach struktury ustalonego typu `str` wraz z procedurami wstawiania i wyszukiwania elementów w drzewie. Zaprogramuj również funkcję `int rozmiar` zwracającą liczbę elementów drzewa. Możliwe są dwie reprezentacje drzew: w jednej z nich wskaźniki do poddrzew są składowymi struktury `str`; w drugiej z nich struktura `str` nie zawiera wskaźników do swoich poddrzew, ale jest „opakowana" w inną strukturę zawierającą jako swoje składowe strukturę `str` oraz wskaźniki. Wybierz dowolną z tych reprezentacji.

Powyższe zadania powinne być zaimplementowane w postaci modułów. Częścią rozwiązania powinien być krótka procedura korzystająca z zaimplementowanego modułu. Przypominam, że oceniane i punktowane są jedynie dwa zadania.

Marcin Młotkowski

16 lutego 2012

Programowanie obiektowe

Lista 2.

Poniższe zadania należy zaimplementować w C#.

Zadanie 1. Zadeklaruj klasę ***IntStream*** implementującą strumień liczb naturalnych, która implementuje metody:

```
int next();
bool eos();
void reset();
```

gdzie kolejne wywołania metody `next()` zwracają kolejne liczby naturalne począwszy od zera, wartość metody `eos()` oznacza koniec strumienia, a `reset()` inicjuje na nowo strumień. Zadeklaruj dwie podklasy

- ***PrimeStream*** implementującą strumień liczb pierwszych, tj. wartościami kolejnych wywołań metody `next()` są kolejne liczby pierwsze. Oczywiście ze względu na ograniczony rozmiar typu `int` możliwe jest jedynie zwrócenie liczb pierwszych mniejszych niż rozmiar typu. Gdy nie jest możliwe obliczenie kolejnej liczby pierwszej, wartość `eos()` powinna być `true`.
- klasę ***RandomStream***, w której metoda `next()` zwraca liczby losowe. W takim wypadku `eos()` jest zawsze fałszywe.

Wykorzystaj te klasy do implementacji klasy ***RandomWordStream*** realizującej strumień losowych słów o długościach równych kolejnym liczbom pierwszym.

Zadanie 2. Zadeklaruj w C# klasę ***Array*** implementującą jednowymiarowe tablice typu `int` za pomocą list dwukierunkowych o początkowych granicach indeksów wskazywanych przez parametry konstruktora. Przyjmij, że rozmiar tablicy i jej granice indeksowania mogą być zmieniane podczas działania programu za pomocą odpowiednich metod. W implementacji zwróć uwagę na to, aby typowe operacje jak

```
Array a1(0,100), a2(0,100), a3(0,100);
for (int i = 0; i < 100; i++)
       a3.set(i, a1.get(i) + a2.get(i));
```

były wykonywane efektywnie, tj. bez przeszukiwania tablicy za każdym razem.

Zadanie 3. Podaj implementację klasy ***BigNum*** pamiętającej duże liczby całkowite (ich maksymalny rozmiar może być zadany stałą) wraz z operacjami dodawania, odejmowania, mnożenia i dzielenia całkowitego. Zaprogramuj również metodę wypisującą liczby na ekranie.

Zadanie 4. Zdefiniuj klasę ***ListaLeniwa*** implementującą leniwą listę liczb całkowitych wraz z metodą

```
int element(int i);
```

zwracającą i-ty element listy. Możesz przyjąć, że elementami tej listy są losowo wybrane liczby całkowite. "Leniwość" takiej listy polega na tym, że na początku jest ona pusta, jednak w trakcie wywołania metody `element(100)` budowanych jest pierwszych sto elementów. Gdy dla takiej listy wywołamy metodę `element(102)` do listy dopisywane są brakujące dwa elementy. Natomiast jeśli teraz zostanie wywołana metoda `element(44)` zostanie odszukany 44. element listy i zwrócony jako wynik.

Zaimplementuj podklasę ***Pierwsze*** zawierającą listę liczb pierwszych, tj. `element(i)` zwraca i-tą liczbę pierwszą.

Za każde zadanie można otrzymać do 4 pkt, jednak można oddać nie więcej niż 2 zadania. Proszę do każdego ocenianego zadania dołączyć króciutki program ilustrujący możliwości zaprogramowanych klas.

Marcin Młotkowski

23 lutego 2012

Programowanie obiektowe

Lista 3.

Zadanie 1. Zaprogramuj klasę ***Lista<T>*** implementującą metody dodawania i usuwania elementów z początku i końca listy, oraz metodę sprawdzania jej niepustości. Istotne jest, aby elementy listy nie były obiektami klasy ***Lista***, lecz elementami innej klasy, której polami są: pole zawierające wartość typu `T`, oraz odnośniki do innych elementów listy. Przyjmij taką implementację klasy ***Lista***, aby działała ona efektywnie zarówno gdy jest wykorzystywana jako kolejka, jak i stos, tj. aby operacje dodawania i usuwania elementów na początek i koniec działały w czasie stałym. Operacja (metoda) usuwania elementu powinna zwracać jako wartość usuwany element.

Zadanie 2. Zaimplementuj klasę ***Słownik<K, V>*** przechowującą pary elementów, gdzie pierwszym elementem pary jest klucz, a drugim wartość. Klasa powinna implementować metodę dodawania, wyszukiwania i usuwania elementu wskazywanego przez klucz.

Zadanie 3. Na wykładzie został omówiony wzorzec `Singleton`, który pozwala na utworzenie tylko jednej instancji klasy. Zaprogramuj klasę, która umożliwia utworzenie tylko co najwyżej N instancji klasy. N niech będzie ustaloną w kodzie źródłowym stałą. Przyjmij, że jeżeli zostanie utworzonych N instancji, to kolejne żądania obiektu zwrócą kolejne istniejące już instancje klasy. Zaprogramuj klasę w wersji leniwej.

Zadanie 4. Zaprogramuj klasę ***Wektor*** implementującą wektory swobodne. Przyjmij, że współrzędne wektora są pamiętane za pomocą liczb typu `float`. Zaprogramuj operatory dodawania wektorów oraz iloczynu skalarnego wektorów i iloczynu wektora przez liczbę.

Korzystając z tej klasy zaprogramuj klasę ***Macierz*** jako tablicę wektorów z operacjami dodawania macierzy, mnożenia i mnożenia przez wektor.

Zastanów się, czy jest możliwa implementacja tych klas jako klas generycznych, tak aby np. można było łatwo implementować wektory lub macierze liczb zespolonych.

Implementacje klas skompiluj w postaci modułów dll. Do każdego zadania dołącz też krótki przykładowy program ilustrujący wykorzystanie zbudowanej biblioteki. Odpowiednie informacje jak to zrobić można znaleźć np. w dokumentacji polecenia `csc` (Windows) lub `mcs` (Mono).

Uwaga 1: Powyższe zadania należy wykonać nie wykorzystując klas bibliotecznych.

Uwaga 2: W środowisku MONO do skompilowania programów generycznych wykorzystuje się polecenie `gmcs`.

Za każde zadanie można otrzymać do 4 pkt, jednak można oddać nie więcej niż 2 zadania. Proszę do każdego ocenianego zadania dołączyć króciutki program ilustrujący możliwości zaprogramowanych klas.

Marcin Młotkowski

1 marca 2012

Programowanie obiektowe

Lista 4.

Zaprogramuj dwa z poniższych zadań.

Zadanie 1. Wybierz zaprogramowane wcześniej przez Ciebie dwie kolekcje. Zastanów się, jakie są wspólne operacje w tych kolekcjach. Zaprogramuj interfejs zawierający nagłówki tych operacji i przebuduj tak implementacje tych kolekcji, aby klasy implementowały ten interfejs. Dodatkowo zaimplementuj w jednej z tych kolekcji interfejs `IEnumerable`.

Zadanie 2. Zaprogramuj jako klasę generyczną skończony bufor przechowujący elementy typu zadanego jako parametr klasy. Rozmiar bufora jest podawany jako parametr konstruktora. Dodawanie elementu powinno być za pomocą operatora "+", a wartością wyrażenia `bufor + element` powinna być wartość logiczna wskazująca że powiodło się dodanie do bufora.

Pobranie elementu powinno być zaprogramowane jako właściwość. Efektem ubocznym pobrania powinno być usunięcie elementu z bufora. Pobierany jest zawsze najdawniej dodany element. W przypadku próby pobrania elementu z pustego bufora program powinien zgłosić wyjątek.

Zadanie 3. Zdefiniuj klasy reprezentujące różne miary czasu wraz z odpowiednimi metodami:

- ***Godzina*** — reprezentuje czas jako trójkę (godzina, minuta, sekunda), oraz metody: `dodaj (int sekundy)` (zwiększa czas o podaną liczbę sekund), `dodaj (Godzina g)` (zwiększa czas o liczbę sekund od północy do g), operator binarny `-(Godzina g1, Godzina g2)` obliczający liczbę sekund pomiędzy czasami, oraz właściwość `Sekundy` obliczającą liczbę sekund od północy. Przyjmij, że godzina to liczba naturalna z przedziału $[0 \ldots 23]$, a minuty i sekundy to liczby z przedziału $[0 \ldots 59]$.

- ***Data*** — (dzień, miesiąc, rok)
  metoda zwiększająca datę o wskazaną liczbę dni: `dodaj (int dni)`, operator `-(Data d1, Data d2)` obliczający liczbę dni dzielącą Daty, oraz własność `Dni` podającą, który to dzień bieżącego roku. Przy dodawaniu dni do daty uwzględnij zasady obowiązujące w kalendarzu gregoriańskim.

Zadanie 4. Zaimplementuj dwie klasy implementujące różne sposoby reprezentacji grafu nieskierowanego; może to być np. reprezentacja macierzowa oraz reprezentacja za pomocą list sąsiedztwa (ale muszą być różne). Przyjmij, że węzły są etykietowane wartościami typu `string`. W każdej klasie zdefiniuj metodę generowania losowego grafu o zadanej liczbie węzłów i krawędzi. Zadeklaruj interfejs zawierający podstawowe metody i własności potrzebne do obsługiwania grafu. Interfejs ten powinien być implementowany przez obydwie klasy.

Następnie zaprogramuj dowolny algorytm wyszukiwania najkrótszej drogi między dwoma węzłami grafu wskazanymi za pomocą etykiet. Zadbaj o to, aby w algorytmie odwoływać się tylko do metod zadeklarowanych w interfejsie. Zmierz czasy wykonania tego algorytmu dla różnych reprezentacji grafu.

Zamiast przeszukiwania możesz też zaimplementować w obydwu klasach odpowiednie metody, dzięki którym grafy staną się prawdziwymi kolekcjami wierzchołków lub krawędzi, które można przetwarzać instrukcją `foreach`. Wybór, czy graf jest kolekcją wierzchołków czy krawędzi należy uzasadnić przy oddawaniu programu.

Zadanie 5. Zaprojektuj i zaimplementuj odpowiedni zbiór klas do reprezentowania produkcji gramatyk bezkontekstowych. Zaimplementuj metodę generowania losowych słów wyprowadzanych w tej gramatyce.

Zadania wykonaj w C#. Tradycyjnie do każdego zadania powinien być dołączony krótki program ilustrujący wykorzystanie zaimplementowanych klas i metod.
Uwaga Powyższe zadania należy wykonać nie wykorzystując klas bibliotecznych. Można jednak korzystać z tablic.

Marcin Młotkowski

8 marca 2012

Programowanie obiektowe

Lista 5.

Poniższe zadania mają być zaimplementowane w Javie. Dla każdego zadań proszę podać krótki program ilustrujący możliwości zaimplementowanych klas.

Zadanie 1. Zaimplementuj kolekcję przechowującą elementy w kolejności *rosnącej* wraz z metodami (lub właściwościami) dodania elementu, pobrania elementu (z jego usunięciem) oraz wypisania wszystkich elementów. Przyjmij, że przy pobieraniu elementu pobierany jest zawsze najmniejszy. Załóż, że elementy przechowywane w tej kolekcji muszą implementować interfejs umożliwiający porównywanie elementów (może to być standardowy interfejs `Comparable<T>`).

Zaimplementuj również dowolną hierarchię klas implementującą interfejs `Comparable<T>` (lub inny zaproponowaną przez Ciebie), zawierającą przynajmniej cztery klasy. Może to być np. hierarchia klas reprezentująca stopnie wojskowe bądź klasy reprezentujące najważniejsze stanowiska Rzeczpospolitej Polskiej (wynik porównania ma odzwierciedlać tzw. precedencję).

Zwróć uwagę, aby implementacja tej hierarchii klas przestrzegała omówionej na wykładzie zasady otwarte–zamknięte, tj. aby można było dodać klasę (np. reprezentującą nowy stopień wojskowy) implementującą `Comparable<T>` bez konieczności zmiany implementacji w pozostałych klasach.

Zadanie 2. Łatwo jest zdefiniować klasę obiektów opisujących automaty skończone rozpoznające język składający się z jednej litery. Mając dane dwa automaty rozpoznające języki L_1 i L_2 łatwo też skonstruować trzeci automat (czyli obiekt odpowiedniej klasy) rozpoznający język będący konkatenacją lub sumą L_1 i L_2. Postępując indukcyjnie możemy skonstruować coraz bardziej rozbudowane automaty. Zaprogramuj klasę (lub hierarchię klas, jeśli jest to wygodniejsze) takich automatów. Podaj implementację klasy (lub metody) rozpoznającej języki z gwiazdką. Jeśli nie wiesz jeszcze jak to zrobić, możesz zaprogramować własny algorytm.

Zadanie 3. Wyrażenia arytmetyczne można reprezentować jako drzewa, gdzie w liściach pamiętane są liczby, a w węzłach symbole operacji arytmetycznych. Zaimplementuj w Javie odpowiednie klasy reprezentujące węzły i liście takiego drzewa jako podklasy klasy ***Wyrażenie***. W każdej klasie zdefiniuj metodę

```
public int oblicz();
```

obliczającą wartość wyrażenia reprezentowanego przez obiekt. Zdefiniuj odpowiednie konstruktory. Przyjmij, że w liściach mogą być zarówno stałe liczbowe jak i zmienne. Przyjmij, że wartości zmiennych są przechowywane np. tablicy haszującej (możesz wykorzystać tu klasy biblioteczne).

Uwaga: nie jest konieczne parsowanie wyrażeń, wyrażenia można budować np. tak:

```
wyrazenie = new Dodaj(new Stala(4), new Zmienna("x"))
```

Zadanie 4. *Zadanie to jest rozszerzeniem poprzedniego zadania.* Podobnie jak wyrażenia możemy też w postaci drzew reprezentować programy. Zaproponuj odpowiednią hierarchię klas, które będą reprezentowały instrukcję przypisania, instrukcję warunkową, instrukcję pętli oraz wypisanie komunikatu na konsolę. Możesz przyjąć, że wyrażenia arytmetyczne można interpretować jako wyrażenia logiczne tak jak w języku `C`.

Jako ilustrację podaj jakiś niebanalny program, np. obliczenie silni.

Marcin Młotkowski

15 marca 2012

Programowanie obiektowe

Lista 6.

Poniższe zadania mają być zaimplementowane w Javie. Dla każdego zadań proszę podać krótki program ilustrujący możliwości zaimplementowanych klas.

Zadanie 1. Wybierz dowolne zadanie z poprzednich list dot. kolekcji (listy, grafy itp) i zaprogramuj je w Javie. Wymuś, aby implementowana kolekcja implementowała interfejs ***Serializable*** (z pakietu `java.io`) tak, aby można było zapisywać i odczytywać kolekcję z pliku dyskowego.

Jako ilustrację programu podaj program który zapisuje kolekcję na dysku a następnie ją odzyskuje.

W internecie jest sporo artykułów opisujących jak implementować ten interfejs. Warto z nich skorzystać.

Zadanie 2. Podobnie jak w poprzednim zadaniu, ale kolekcja winna implementować interfejs ***Collection<E>*** z pakietu `java.utils`.

Poszukaj informacji, jakie korzyści daje implementacja tego interfejsu i zaprezentuj te korzyści w przykładach.

Zadanie 3. Zaprogramuj klasę implementującą dostęp do bufora o stałym rozmiarze przechowującym elementy typu generycznego `T`. Implementacja powinna umożliwić działanie takiego bufora w środowisku wielowątkowym. Kolejność elementów pobieranych z bufora powinna być taka sama jak kolejność ich wkładania do bufora.

Korzystając z tej klasy zaimplementuj problem *producenta–konsumenta*: producent produkuje wyniki (napisy) i wkłada je do bufora. Jeśli bufor jest pełny, to producent zasypia czekając aż zwolni się miejsce w buforze. Konsument, jeśli w buforze jest jakiś element (napis), to go pobiera i "konsumuje". Zaimplementuj producenta i konsumenta jako dwa odrębne wątki.

Zadanie 4. Wiele zadań programistycznych ma swoje naturalne rozwiązania w postaci *potoku* procesów, gdzie każdy proces wykonuje pewien fragment zadania i wynik przekazuje do następnego procesu wykorzystując np. taki bufor jak w zadaniu poprzednim. Przykładem takiego zadania jest problem odfiltrowania (wg. wskazanego kryterium) i posortowania alfabetycznie listy nazwisk: jeden proces pracuje w cyklu: pobierz wiersz, filtruj, wyślij do bufora; zaś drugi pobiera kolejne wiersze z bufora i wstawia je w odpowiednie miejsce i na końcu wypisuje je na konsolę. Zaimplementuj za pomocą wątków takie zadanie wykorzystując jako kanał komunikacyjny implementację bufora z poprzedniego zadania.

Zadanie 5. Algorytm sortowania tablicy elementów przez *scalanie* działa następująco: najpierw tablica jest dzielona na pół. Następnie każda z tych mniejszych tablic jest porządkowana. Na końcu obydwie posortowane tablice są scalane. Zaprogramuj sortowanie przez scalanie tablicy elementów `int` tak, aby operacje sortowania podtablic były odrębnymi wątkami.

Marcin Młotkowski

23 marca 2012

Programowanie obiektowe

Lista 7.

Zadanie polega na implementacji edytora obiektów. Na początek proszę o zadeklarowanie w Javie prostej klasy oraz jej dwóch podklas. Mogą to być (do wyboru):

- klasa ***Książka*** wraz z podklasami ***WydawnictwoCiągłe*** i ***Czasopismo***;
- klasa ***Figura*** wraz z podklasami ***Okrąg*** i ***Trójkąt***;
- klasa ***Pojazd*** wraz z podklasami ***Samochód*** i ***Tramwaj***.

Każda klasa powinna implementować przynajmniej 3 pola; wystarczy tylko jedna metoda, np. `toString()`.

Zaimplementuj dla każdej zadeklarowanej klasy:

interfejs do edycji (6 pkt) taki interfejs wygodnie jest zaimplementować jako kontrolkę Swinga, tj. podklasę jednej z podklas `JComponent` (lub pochodnej[1]); wtedy taką własną kontrolkę można w przyszłości umieszczać w oknie wraz z innymi kontrolkami;

intefejs `Serializable` (2 pkt) implementacja tego interfejsu umożliwia zapis i odczyt obiektu do pliku. Zaprogramuj też metody odczytu/zapisu obiektu z/do pliku dyskowego.

Jako ilustrację napisz krótki program uruchamiający z linii poleceń edycję obiektów zapisanych w plikach. Argumentami wywołania programu są: nazwa pliku w którym przechowywany jest pojedynczy obiekt, zaś drugim argumentem jest nazwa klasy obiektu. Jeśli plik nie istnieje, to tworzony jest nowy obiekt.

Marcin Młotkowski

[1] dobrym kandydatem może być np. ***JPanel***

29 marca 2012

Programowanie obiektowe

Lista 8.

Poniższa lista zadań jest do zrobienia w języku Ruby. Każde zadanie to 4 punkty. Wybierz 2 zadania.

Zadanie 1. Rozszerz standardową klasę ***Fixnum*** o metody:

- zeroargumentową metodę `prime?` sprawdzającą pierwszość liczby, tj. wywołanie `5.prime?` ma zwrócić `true`, zaś `6.prime?`: `false`;
- jednoargumentową metodę `ack(y)` obliczającą funkcję Ackermanna zdefiniowaną następująco:

$$Ack(n,m) = \begin{cases} m+1 & \text{gdy } n = 0 \\ Ack(n-1,1) & \text{gdy } m = 0 \\ Ack(n-1, Ack(n,m-1)) & \text{w pozostałych przypadkach} \end{cases}$$

  Na przykład `2.ack(1)` powinno dać 5. Uwaga: funkcja ta bardzo długo liczy, nawet dla niedużych argumentów, więc nie testujcie jej na dużych (> 2) liczbach;
- zeroargumentowa metoda `doskonala`, która zwraca `true` gdy liczba jest doskonała[2];
- zeroargumentową metodę zamieniającą liczbę na jej postać słowną. Można przyjąć, że postać słowna jest uproszczona, np. `123.slownie` powinno zwrócić "jeden dwa trzy".

Zadanie 2. Zaimplementuj dwie klasy: ***ImageBW*** i ***ImageC*** implementujące odpowiednio bitmapowe obrazy czarno-białe i kolorowe. Rozmiary obrazów są ustalane przy tworzeniu obiektów jako parametry konstruktora. Zaprogramuj w tych klasach metody `+(arg)`, `*(arg)`, które zwracają nowy obiekt klasy ***ImageBW*** bądź ***ImageC***. Metoda `+(arg)` tworzy nowy obraz, którego każdy piksel jest alternatywą bitową odpowiednich piksli obiektu i argumentu; odpowiednio `*(arg)` oznacza utworzenie nowego obiektu z koniunkcji bitów piksli. Takie operacje mają sens, gdy operacje wykonujemy na obrazkach o tych samych rozmiarach i tym samym typie (tj. tylko kolorowe z kolorowymi albo czarno–białe z czarno–białymi). Można przyjąć, że zawsze wykonujemy metody na poprawnych danych. Dodaj też do tych klas metodę `narysuj` rysującą obrazy w postaci ascii-artu.

Zadanie 3. Jedną z najprostszych metod szyfrowania jest szyfr podstawieniowy, w którym za literę podstawia się inną literę znajdującą się o K pozycji dalej w alfabecie (Cezar używał tego szyfru dla $K = 3$). Na przykład napis 'Ala ma kota' dla $K = 3$ jest zamieniany na coś w rodzaju 'Dod#pd#nrwd'. Zaprogramuj dwie klasy:

- klasę ***Jawna*** przechowującą napis w postaci jawnej i implementującą metodę `zaszyfruj(klucz)` zwracającą obiekt klasy ***Zaszyfrowane***;
- klasę ***Zaszyfrowane*** przechowującą napis zaszyfrowany i implementującą metodę `odszyfruj(klucz)` zwracającą obiekt klasy ***Jawna***.

Obydwie klasy winne implementować metodę `to_s`. Argument `klucz` to oczywiście długość przesunięcia K.

Marcin Młotkowski

[2] definicję można znaleźć m. in. w Wikipedii

5 kwietnia 2012

Programowanie obiektowe

Lista 9.

Zadanie 1. Bloki jednoparametrowe można traktować jak definicję jednoargumentowej funkcji matematycznej. Na przykład blok { | `x` | `x*x*Math.sin(x)` } reprezentuje funkcję $x \to x^2 * \sin(x)$. Dzięki temu można zdefiniować własną klasę ***Funkcja*** reprezentującą funkcje, gdzie definicja funkcji jest zadana blokiem (a właściwie obiektem klasy ***Proc***) w konstruktorze.

Zaimplementuj klasę ***Funkcja***[3] wraz z metodami

- `.value(x)` oblicza wartość funkcji w punkcie x;
- `.zerowe(a,b, e)` oblicza miejsca zerowe funkcji w przedziale $[a, b]$ z dokładnością e lub zwraca *nil* jeśli miejsce zerowe nie zostało znalezione;
- `.pole(a,b)` oblicza przybliżone pole powierzchni między wykresem a osią OX w przedziale $[a, b]$ (czyli całkę oznaczoną ;). Można tu przyjąć, że wykres jest zawsze nad osią OX;
- `.poch(x)` oblicza wartość (przybliżoną) pochodnej w punkcie x.

Zadanie 2. Zadanie jest podobne do poprzedniego, ale tym razem chcemy reprezentować funkcje dwuargumentowe za pomocą obiektów klasy ***Funkcja2***. Zaprogramuj taką klasę wraz z metodami:

- `.value(x, y)` oblicza wartość funkcji w punkcie (x,y);
- `.objetosc(a, b, c, d)` oblicza przybliżoną objętość między wykresem funkcji a leżącym na płaszczyźnie $OXOY$ prostokątem $[a, b] \times [c, d]$;
- `.poziomica(a,b,c,d,wysokosc)` oblicza listę par (x, y) takich że *f.value(x,y)* $\approx$ *wysokosc*, przy czym $a \leq x \leq b$ oraz $c \leq y \leq d$. Dokładność przybliżenia do *wysokosc* może być zadana w metodzie, podobnie jak dokładność wyszukiwania poziomicy.

Zadanie 3. Rozszerz implementację jednej z klas ***Funkcja*** lub ***Funkcja2*** o metodę rysującą wykres funkcji w zadanym przedziale. Wynikiem działania tej metody może być bądź szkic wykresu zrobiony za pomocą znaków ASCII na konsoli, bądź też plik z bitmapą wykresu, do obejrzenia w jakimś programie graficznym; stosunkowo łatwo będzie skorzystać z bitmapowego formatu PBM lub PPM.

Dla ***Funkcja2*** wykresem może być narysowane poziomice; takie jak np. na mapach fizycznych czy topograficznych.

Marcin Młotkowski

[3] można alternatywnie rozszerzyć istniejącą klasę

12 kwietnia 2012

Programowanie obiektowe

Lista 10.

Poniższa lista zadań jest do zrobienia w Ruby. Każde zadanie to 4 punkty. Wykonaj dwa z tych zadań.

Zadanie 1. Algorytmy sortowania kolekcji wymagają kilku elementarnych operacji na kolekcji takich jak `swap(i, j)` zamieniające miejscami elementy, `length()` zwracająca długość kolekcji, czy `get(i)` zwracająca i-ty element kolekcji.

Zaimplementuj dwie klasy:

1. ***Kolekcja*** implementująca dowolną kolekcję wraz z wyżej wymienionymi metodami (ew. jeszcze dodatkowe jeśli będzie taka potrzeba);
2. ***Sortowanie*** implementującą dwa wybrane algorytmy (lub *strategie*) sortowania kolekcji. Klasa ta powinna implementować metody `sort1(kolekcja)`i `sort2(kolekcja)`, których argumentem jest obiekt klasy ***Kolekcja***. Która metoda jest szybsza?

Ważne w tym zadaniu jest to, aby metoda korzystała wyłącznie z metod udostępnianych przez kolekcję i nie odwoływała się w żaden sposób do implementacji ***Kolekcji***.

Czy potrafisz powiedzieć, jaki to *wzorzec projektowy*?

Zadanie 2. Zaprogramuj klasę ***Kolekcja*** implementującą kolekcję przechowującą elementy dowolnego typu w kolejności rosnącej. Lista ta powinna być zaprogramowana jako lista dwukierunkowa. Zaprogramuj również klasę ***Wyszukiwanie*** wraz z dwiema różnymi metodami wyszukiwania elementu w kolekcji. Dobrymi kandydatami są tu: algorytm wyszukiwania binarnego oraz jego wariant: wyszukiwanie interpolacyjne. Podobnie jak w poprzednim zadaniu, kolekcja powinna być argumentem wywołania metody, a metoda powinna być tak napisana, aby nie korzystała ze szczegółów implementacji kolekcji.

Zadanie 3. Zadanie polega na implementacji klasy (a właściwie kilku klas) służącej do zapamiętania w czytelnej postaci chwilowego stanu programu, tj. stanu obiektów istniejących w systemie. Chcemy, aby była możliwość zapisania stanu w pliku zarówno w formacie html, jak i LaTeX'a (zamiennie: w miarę czytelnie sformatowany plik tekstowy).

Zadanie można podzielić na dwie części:

1. analiza stanu programu
2. generowanie sformatowanego tekstu

Zaprogramuj klasę ***Snapshot*** z metodą `run` analizującą stan programu oraz klasy ***FormatHTML*** i ***FormatTXT*** implementujące metody formatujące. Zadanie można zaprogramować na dwa sposoby: albo klasy ***FormatHTML*** i ***FormatTXT*** są podklasami ***Snapshot***, albo obiekty klas ***FormatHTML*** czy ***FormatTXT*** są argumentami metody `run`. Przy oddawaniu programu oceń wady i zalety poszczególnych rozwiązań.

To już ostatnia lista zadań. Na stronie są już propozycje projektów programistycznych, ale można oczywiście zaproponować własne.

Marcin Młotkowski