ZSI. Egzamin z NMJP – Programowanie Obiektowe (23.III.2002)

Zad 1 (25 p.)

W grze w tysiąca bierze udział 3 graczy. Gracze grają talią składającą się z 24 kart: 9-tki, walety, damy, króle, dziesiątki i asy (karty podano wg rosnącej siły, tzn. as bije wszystkie inne karty tego samego koloru, dziesiątki wszystkie inne karty poza asami itp.). Gra składa się z kolejek i toczy się aż do zdobycia przez (co najmniej) jednego z graczy 1000 lub więcej punktów (stąd nazwa). Wygrywa ten gracz, który w momencie zakończenia gry ma najwięcej punktów. Każda kolejka składa się z:

1. Potasowania i rozdania kart.
2. Licytacji.
3. Rozgrywki (szczegółowe zasady przeprowadzania rozgrywki nie interesują nas w tym zadaniu).

Na początku każdej kolejki jeden z graczy tasuje i rozdaje karty. Karty rozdawane są następująco: każdy z graczy dostaje po 7 kart, 3 karty zostają (zakryte) na stole.

Podczas licytacji gracze kolejno deklarują, ile oczek podejmują się zdobyć podczas rozgrywki (ugrać) lub pasują (oczywiście należy podawać większe liczby oczek niż poprzedni gracz). Licytację wygrywa ten z graczy, który zalicytuje największą liczbę oczek. Licytacja kończy się po dwu kolejnych pasach. W każdej kolejce jeden wybrany gracz musi zacząć licytowanie od 100 oczek (niezależnie od tego, czy ma odpowiednio silne karty). Dzięki temu zawsze jest określony zwycięzca licytacji. Zwycięzca licytacji bierze trzy karty leżące na stole, następnie daje po jednej karcie (nie koniecznie z tych trzech ze stołu) obu partnerom. Każda karta ma określoną liczbę oczek (nie jest istotne w tym zadaniu jaką). Ponadto, jeśli gracz ma króla i damę tego samego koloru (co nazywa się w tej grze meldunkiem), może w czasie rozgrywki wychodząc w jedną z tych kart (ale tylko wychodząc, a nie dokładając do karty wychodzącego) zgłosić meldunek, za co dostaje dodatkowe punkty (znów w tym zadaniu nie jest ważne ile). Zatem gracz może mieć meldunek i nie dostać za niego w czasie rozgrywki punktów, jeśli nie uda mu się zgłosić tego meldunku – bardzo częsta sytuacja, gdy inny gracz ma np. asa i dziesiątkę w kolorze meldunku.

Należy zaprojektować w **Smalltalku** klasy służące do przeprowadzania symulacji gry w 1000. W projekcie należy uwzględnić wszystkie atrybuty i metody potrzebne do tasowania kart, rozdawania kart i przeprowadzania licytacji (czyli bez uwzględniania rozgrywki). Należy wyróżnić (co najmniej) klasy: **Karta**, **Gra** i **Gracz**.
Należy też przewidzieć następujące rodzaje graczy:

- **losowy**: z prawdopodobieństwem 0.5 zgłasza liczbę oczek o 10 większą od dotąd wylicytowanej,
- **optymista**: jeśli suma oczek, które na pewno uda mu się ugrać bez meldunków plus suma punktów za wszystkie posiadane meldunki jest większa bądź równa od dotąd wylicytowanej liczby oczek + 10, to zgłasza liczbę oczek większą od dotąd wylicytowanej o 10, wpp. pasuje,
- **ostrożny**: jeśli suma oczek, które na pewno uda mu się zdobyć bez meldunków plus suma punktów za te posiadane meldunki, do których ma (w tym samym kolorze) asa i dziesiątkę jest większa bądź równa od dotąd wylicytowanej liczby oczek + 10, to zgłasza liczbę oczek większą od dotąd wylicytowanej o 10. wpp. pasuje.

(Powyższe zasady nie dotyczą oczywiście rozpoczynania licytacji, tam każdy graczy musi zalicytować 100).

Opisz atrybuty i metody, podaj związki między klasami.

Zaimplementuj dla graczy metody:

- **`rozdaj: talia dla: gracz1 oraz: gracz2`**
  tasuje karty z talii i rozdaje je (wg zasad) sobie i dwu pozostałym graczom, jako wynik daje kolekcję (**Set**) trzech pozostałych po rozdaniu kart.
- **`ileLicytujesz: aktualnaLiczbaOczek`**
  zgodnie z rodzajem gracza, na podstawie jego kart i zadanej jako parametr dotąd wylicytowanej liczby oczek zgłasza nową (wyższą) liczbę oczek (wynik metody) lub pasuje (wtedy wynikiem metody jest **nil**). Ta metoda nie jest wywoływana na samym początku licytacji (bo wtedy i tak rozpoczynający gracz musi zadeklarować 100).

Należy zaimplementować wszystkie niestandardowe metody użyte w implementacji powyższych dwu metod, poza:

- metodą gracza **`ileOczekWeźmieszNaPewno`** (jako wynik daje liczbę oczek, które gracz ugra na pewno bez meldunków).
- metodą gry **`punktyZaMeldunek: karta`** (jako wynik daje liczbę punktów przyznawaną za zgłoszenie meldunku w kolorze podanej jako parametr karty, karta powinna być królem albo damą).

Zad 2 (15 p.)

Dane są deklaracje:

```
struct Osoba { // Informacje o osobie zapisane w książce telefonicznej
  char imie[MAX_IMIE],nazwisko[MAX_NAZWISKO],telefon[MAX_TELEFON];
};

class PKT { // Prywatna Książka Telefoniczna
public:
  virtual char* operator[](char*);
          // Zwraca numer telefonu osoby o podanym nazwisku (gdy jest takich wiele, zwraca numer dowolnej
          // z nich) lub NULL, jeśli osoby takiej nie ma w książce telefonicznej
  virtual PKTIterator* select(Osoba&)=0;
          // Tworzy iterator umożliwiający dostęp do informacji o osobach zapisanych w książce telefonicznej.
          // Argument metody pełni podwójną rolę:
          //
          // * określa, jakie osoby nas interesują – umieszczając w polu struktury przekazanej jako argument
          //   napis niepusty ustalamy, że chcemy otrzymać wyłącznie informacje o osobach, dla których ten
          //   atrybut (imię, nazwisko lub telefon) ma podaną wartość. Np. przekazując jako argument select
          //   strukturę mającą w polu 'imie' napis "Jan" a w pozostałych polach napisy puste ("") prosimy
          //   o iterator dający dostęp do informacji o wszystkich Janach.
          //
          // * jest przez iterator traktowany jako bufor, za pośrednictwem którego zwracane są kolejne
          //   struktury
          //
          // Po utworzeniu iterator wpisuje do struktury przekazanej jako argument tej metody informacje o
          // pierwszej znalezionej osobie (chyba, że takiej nie ma) i umożliwia badanie, czy jesteśmy już na
          // końcu (co oznacza, że „bufor" nie zawiera już informacji o osobie) oraz przechodzenie do
          // następnej osoby (wolno to robić tylko, jeśli nie jest jeszcze koniec)
  virtual PKT& operator+=(PKT&);
          // Dopisuje do książki wszystkie informacje pobrane z książki przekazanej jako argument operatora
  virtual PKT& operator+=(Osoba&)=0;
          // Dodaje do książki informacje o podanej osobie (nie sprawdza, czy jest już ktoś o takim nazwisku)
  … // Inne potrzebne elementy
};

class PKTIterator { // Iterator dla książki telefonicznej.
                    // Iteratory dla konkretnych implementacji dziedziczą z tej klasy
public:
  virtual void next()=0;
               // Przejście do następnej osoby
  virtual int atEnd()=0;
               // Sprawdzenie, czy jesteśmy na końcu
  … // Inne potrzebne elementy
};
```

a) Zaimplementuj te metody klasy **PKT**, które nie są abstrakcyjne.
b) Uzupełnij definicję **PKT** o elementy brakujące do tego, by obiekty realizujące książkę telefoniczną można było tworzyć, niszczyć, przypisywać i kopiować.
c) Zdefiniuj operator wypisywania zawartości książki telefonicznej.
d) Zdefiniuj klasę **PKTTab** realizującą książkę telefoniczną przy pomocy tablicy. Zapisz wszystkie potrzebne metody tej klasy. **Uwaga: nie trzeba implementować iteratora dla PKTTab.**