Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Objektno orijentisano programiranje
(OF2OO1, OS2OO1, OS3OOP, OE2OOP, OT2OOP)
Nastavnik: doc. dr Dragan Milićev
Ispitni rok: Januar 2007.
Datum: 12.2.2007.

Kandidat: _____

Broj Indeksa: _____ *E-mail:*_____

Ispit traje 3 sata. Dozvoljeno je korišćenje literature.

Pismeni ispit:		***Domaći zadaci:***	
Zadatak 1	_____/10	*Obavezni deo*	_____/40
Zadatak 2	_____/10	*Neobavezni deo*	_____/10
Zadatak 3	_____/10		
Zadatak 4	_____/10		
Zadatak 5	_____/20		
Ukupno na ispitu:	**_____/60**	***Ukupno na domaćem:***	**_____/50**

Ukupno: _____/110

Ocena: _____ (_____)

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Ocenjivanje unutar potpitanja je po sistemu "sve ili ništa", odnosno nema parcijalnih poena. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**. Zadaci 1-4 su eleminatorni prema Pravilima predmeta.

1. (10 poena)

Dat je deo koda na jeziku C++. Za svaku liniju označenu brojevima 1-5 navesti da li je ispravna (prevodilac neće prijaviti grešku) ili nije (prevodilac će prijaviti grešku) – u srednju kolonu upisati „Da“ ili „Ne“. Ako linije nije ispravna, u trećoj koloni kratko navesti vrstu greške (uzrok).

```
void main () {
  int a[10];
  int* p=a; int* q;
  q  = &*p++;   // 1
  &q = *p++;    // 2
  *q = *p++;    // 3

  extern int f();
  void (*pf1)(int) = &f;  // 4
  int (*pf2)() = &f;      // 5
}
```

Odgovor:

Linija	Ispravna	Vrsta (uzrok) greške
1	DA	
2	NE	1. Type mismatch: cannot convert from 'int' to 'int**' 2. Left operand of operator = (&q) is not an lvalue.
3	DA	
4	NE	1. Type mismatch: cannot convert from 'int (*)()' to 'void (*)(int)'
5	DA	

2. (10 poena)

Dat je sledeći program:

```
#include <iostream.h>

class Object {
public:
  Object (int i, Object* nxt=0) : myID(i), next(nxt) {}
  virtual ~Object () { if (next) delete next; }

  virtual void write();

protected:
  int myID;
  Object* next;
};

void Object::write () {
  if (next) next->write();
  cout<<", "<<myID;
}

class Comedian : public Object {
public:
  Comedian (int i, Object* nxt=0) : Object(i,nxt) {}
  virtual void write();
};

void Comedian::write () {
  cout<<myID;
}

void main () {
  Object* p = new Object(1,new Object(2,new Comedian(3,new Object(4,new
Comedian(5)))));
  p->write();
  delete p;
  p = new Object(3,new Object(2,new Object(1)));
  p->write();
  delete p;
}
```

Šta ispisuje ovaj program (odgovor ispisati tačno kako i program ispisuje)?

Odgovor:

3, 2, 1, 1, 2, 3

Objašnjenje:

`Object* p = new Object(1,new Object(2,new Comedian(3,new Object(4,new Comedian(5)))));`

pravi listu (`Object`, `Object`, `Comedian`, `Object`, `Comedian`) sa vrednostima 1,2,3,4,5.

`p->write` poziva rekurzivno polimorfne funkcije, pri čemu objekat tipa `Object` najpre poziva istu funkciju sledećeg u listi pa onda ispisuje svoj podatak, dok `Comedian` prekida rekurziju i samo ispisuje svoj podatak, pa će ispis biti "**3, 2, 1**".

Slično,

`p = new Object(3,new Object(2,new Object(1)))`

pravi listu od 3 objekta klase `Object` sa vrednostima 3, 2, 1 redom.

`p->write()` ispisuje "**, 1, 2, 3**".

3. (10 poena)

Na jeziku C++ realizovati klasu `Vector` koja apstrahuje vektor u Dekartovom koordinatom sistemu. Vektor se predstavlja tačkom `(x,y)` (vektor uvek polazi iz koordinatog početka), a koordinate su tipa `double`. Ova klasa treba da obezbedi sledeće:

- Inicijalizaciju vektora koordinatama, pri čemu je podrazumevana inicijalizacija na (0,0).
- Operacije poređenja na jednakost (`operator==`) i nejednakost (`operator!=`) dva vektora.
- Operacije sabiranja (`operator+`) i oduzimanja (`operator-`) dva vektora.
- Operaciju skalarnog proizvoda dva vektora (`operator*`).
- Operaciju množenja skalara i vektora (`operator*`).

Rešenje:

```
class Vector {
public:

	Vector(double xx=0, double yy=0) : x(xx), y(yy) {}

	friend bool operator== (const Vector& v1, const Vector& v2)
	{
		return v1.x == v2.x && v1.y == v2.y;
	}

	friend bool operator!= (const Vector& v1, const Vector& v2)
	{
		return !(v1 == v2);
	}

	friend Vector operator+ (const Vector& v1, const Vector& v2)
	{
		return Vector(v1.x + v2.x, v1.y + v2.y );
	}

	friend Vector operator- (const Vector& v1, const Vector& v2)
	{
		return Vector(v1.x - v2.x, v1.y - v2.y );
	}

	friend double operator* (const Vector& v1, const Vector& v2)
	{
		return v1.x * v2.x + v1.y * v2.y ;
	}

	friend Vector operator* (const Vector& v, double s)
	{
		return Vector(v.x * s, v.y * s );
	}

	friend Vector operator* (double s, const Vector& v)
	{
		return v*s;
	}

private:
	double x, y;
};
```

4. (10 poena)

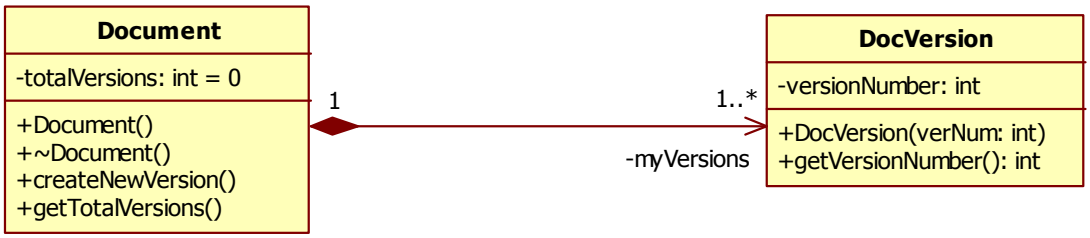
Potrebno je realizovati sledeći sistem klasa i njihovih relacija. *Dokument* (*Document*) može imati jednu ili više svojih *Verzija* (*DocumentVersion*). Prilikom kreiranja novog Dokumenta, kreira se njegova početna Verzija. Prilikom uništavanja Dokumenta, uništavaju se sve njegove Verzije. Svaka verzija ima svoju oznaku koja predstavlja redni broj te Verzije: kada se kreira nova Verzija Dokumenta, ovaj broj se uvećava za jedan.

(a) Nacrtati dijagram klasa koji prikazuje UML model opisanog sistema.

(b) Napisati kompletan C++ kod klase Dokument (*Document*) i svih njenih operacija, uključujući operaciju `createNewVersion()`.

Rešenje:

a)



b)

```
class DocVersion
{
public:
      DocVersion(int verNum) : versionNumber(verNum) {      }
      int getVersionNumber() { return versionNumber; }

private:
      int versionNumber;
};

class Document
{
public:
      Document();
      virtual ~Document();
      void createNewVersion();
      int getTotalVersions();

private:
      // Kolekcija verzija dokumenta je obican dinamicki niz;
      // dinamicki niz se moze zameniti bilo kojom kolekcijom
      // (CollectionU, std::vector, std::list, ...)
      int totalVersions;
      DocVersion **myVersions;

      // Pomocna metoda za dodavanje verzije u kolekciju verzija
      void addVersionToVersionCollection(DocVersion *pDocVersion);
      // Pomocna metoda za unistavanje svih verzija
      void destroyAllVersions();
};

Document::Document() : totalVersions(0), myVersions(0)
{
      createNewVersion();
}
```

```
Document::~Document()
{
	destroyAllVersions();
}

void Document::createNewVersion()
{
	// Stvara se nova verzija...
	DocVersion *pDocVersion = new DocVersion(totalVersions);
	// ... i dodaje u kolekciju verzija
	addVersionToVersionCollection(pDocVersion);
}

int Document::getTotalVersions()
{
	return totalVersions;
}

void Document::addVersionToVersionCollection(DocVersion *pDocVersion)
{
	DocVersion **temp = new DocVersion*[totalVersions+1];
	if (temp) {
		for(int i=0; i<totalVersions; i++) {
			temp[i] = myVersions[i];
		}
		temp[totalVersions] = pDocVersion;
		totalVersions++;
		delete [] myVersions;
		myVersions = temp;
	}
}

void Document::destroyAllVersions()
{
	// Unistavanje verzija dokumenta
	for(int i=0; i<totalVersions; i++)
		delete myVersions[i];
	// Dealokacija niza pokazivaca na verzije
	delete [] myVersions;
}
```

5. (20 poena) Konstruktivni zadatak

Radi optimalne organizacije javnog skupa, potrebno je napraviti uprošćenu simulaciju potrošnje piva. Pri realizaciji sistema, poći od sledećih pretpostavki:

- Šanker je sve vreme koncerta zaposlen i konstantno toči pivo, koje raspoređuje na N mesta na šanku. Za točenje svakog piva potrebno je T (±10%) sekundi (ravnomerna raspodela). Mesta se opslužuju kružnim redosledom, počev od prvog.
- Na svakom od mesta se sigurno nalazi posetilac javnog skupa, koji konzumira pivo, koje se zadržava u organizmu posetioca Z (±30%) minuta, pre nego posetilac ode do sanitarnog čvora i izluči pivo. Ako posetilac u nekom trenutku u organizmu ima više od M piva, odlazi do sanitarnog čvora i izbacuje sva piva koja ima u organizmu.
- Zadržavanje posetioca u sanitarnom čvoru je isto za sve posetioce.

N i T, odnosno Z i M, ne moraju biti isti za sve šankere, odnosno posetioce. Potrebno je projektovati klase koje će podržati opisano ponašanje. Nije potrebno pisati programski kod koji stvara i povezuje opisane objekte. Pri ovome, može se koristiti i/ili menjati postojeći programski kod TSS.

a) (7) Šanker.

b) (5) Posetilac (osim izbacivanja svih piva).

c) (5) Posetilac (izbacivanje svih piva).

d) (3) Sanitarni čvor.

Rešenje:

a)

Sanker.h

```
#ifndef Sanker_h
#define Sanker_h 1

#include "Interfaces.h"
#include "AbstractConcepts.h"
#include "UniRndGen.h"

class Visitor;
class Posetilac;

// Klasa Sanker implementira IFlowSource;
// Sanker jer je izvor (piva) i nasledjuje RandomTimedFlowlElement
// jer Scheduler treba da kontrolise tocenje piva
class Sanker : public IFlowSource, public RandomTimedFlowElement
{
public:
      // Inicijalizacija promenljivih kao i inicijalizacija
      // generatora slucajnih brojeva (po tekstu zadatka, uniformni)
      Sanker (int NN, double TT)
            : N(NN), T(TT), RandomTimedFlowElement(new UniformGenerator (0.9*T,
1.1*T))
      {
            // Niz od N posetilaca (Posetilac je klasa uradjena pod b) i c))
            pos = new Posetilac*[N];
            for (int i=0; i<N; i++)
            {
                  pos[i] = 0;
            }
            // Pocinje se od posetioca sa rednim brojem 0
            trenutniBrPos = 0;
      }

      ~Sanker ()
      {
```

```
            delete [] pos;
      }

      bool addPosetilac(Posetilac*);

      virtual void accept (Visitor* vis);

      virtual void notify (int ID);

      // Ukljucivanje Sankera (radi)
      inline void switchOn () { onOff = true; }

      // Iskljucivanje Sankera (ne radi)
      inline void switchOff () { onOff = false; }

      // Ispitivanje da li radi/ne radi
      inline bool isOn () { return onOff; }

protected:
      int N;                  // max. broj posetilaca
      double T;               // srednje vreme izmedju dva tocenja piva
      int trenutniBrPos;      // redni broj trenutnog posetioca
      Posetilac **pos;        // dinamicki niz posetilaca
      bool onOff;             // flag koji kaze da li sanker radi ili ne
};

#endif
```

Sanker.cpp

```
#include "Sanker.h"
#include "Posetilac.h"

void Sanker::accept (Visitor* vis)
{
      if (vis) vis->visitSanker(this);
}

void Sanker::notify(int ID)
{
      // Kada Sanker dobije notify od Scheduler-a,
      // treba da nadje prvog sledeceg aktivnog
      // posetioca i da mu "sipa" pivo preko acceptFlow()
      int i = trenutniBrPos;
      int brojac=0;
      while ((pos[i]==0) && (brojac<N))
      {
            i++; i=i%N; brojac++;
      }

      // Ako uopste ima Posetioce
      if (brojac<N)
      {
            pos[i] ->acceptFlow();
            trenutniBrPos = i++;
            trenutniBrPos = trenutniBrPos % N;
      }
      // i nakon toga da se opet prijavi Scheduler-u
      // preko raiseEvent() za sledece sipanje piva
      raiseEvent ();
}

bool Sanker::addPosetilac(Posetilac* p)
{
      int i=0;
      while ((pos[i]==0) && (i<N))
```

```
            i++;

      if (i<N)
      {
            pos[i] = p;
            return true;
      }
      else
            return false;
}
```

b)

Posetilac.h

```
#ifndef Posetilac_h
#define Posetilac_h 1

#include "AbstractConcepts.h"
class Visitor;

// Klasa Posetilac nasledjuje RandomTimedFlowElement
// jer zadrzavanje piva u njemu treba da kontrolise Scheduler
class Posetilac : public RandomTimedFlowElement
{
public:
      // Inicijalizacija promenljivih kao i inicijalizacija
      // generatora slucajnih brojeva (po tekstu zadatka, Uniformni)
      Posetilac (double ZZ, int MM)
      : Z(ZZ), M(MM), RandomTimedFlowElement(new UniformGenerator (0.9*Z*60,
1.1*Z*60))
      {
            brojPiva = 0;  // posetilac u pocetku ima 0 piva u sebi
      }

      // Povezivanje sa Visitor klasama
      virtual void accept (Visitor* vis);

      virtual void acceptFlow();

      // Scheduler preko ove f-je obavestava posetioca
      virtual void notify (int ID);

protected:
      int brojPiva;      // trenutni broj piva
      int M;             // maksimalan broj dozvoljenih piva
      double Z;          // prosecno vreme zadrzavanja jednog piva
};

#endif
```

Posetilac.cpp

```
#include "Posetilac.h"

#include "Visitor.h"
#include "UniRndGen.h"

void Posetilac::accept (Visitor* vis)
{
      if (vis) vis->visitPosetilac (this);
}

void Posetilac::acceptFlow()
{
      // Kada dobije pivo, broj piva se uvecava za 1
      // i prijavljuje se Event Sheduler-u
      // za vreme koje vrati Uniform generator
```

```
	// za odlazak u Sanitarni Cvor
	brojPiva++;
	raiseEvent ();
}

void Posetilac::notify (int ID)
{
	// Kada Scheduler obavesti Posetioca, on odlazi do Sanitarnog cvora
	// gde se izvrsava izbacivanje jednog piva
	if (target)
	{
		send();
		brojPiva--;
	}
}
```

c) Treba izmeniti funkciju `Posetilac::acceptFlow()`, tako da kad uzme pivo više nego što sme, posetilac izbaci sva piva i pozove `Scheduler` tako da ukloni iz svoje liste sve događaje koji se odnose na posetioca (sva piva koja je posetilac popio).

Posetilac-c.cpp

```
#include "Visitor.h"
#include "UniRndGen.h"
#include "Sched.h"

#include "Posetilac.h"

// Za resavanje zadatka pod c) potrebno je:
// prepraviti metodu acceptFlow
// i dodati u scheduler metodu (ili napraviti novu klasu
// koja nasledjuje scheduler i ima metodu za)
// otklanjanje svih objekata tipa event za datog posetioca
// (koji premasi broj od M piva i izbaci ih sve odjednom)

void Posetilac::accept (Visitor* vis)
{
	if (vis) vis->visitPosetilac (this);
}

void Posetilac::acceptFlow()
{
	// Kada dobije pivo, broj piva se uvecava za 1;
	// prijavljuje se dogadjaj za Scheduler u normalnom slucaju;
	// ako je broj piva premasio M, sva piva se odmah izbacuju

	brojPiva++;
	if (brojPiva <= M )
		raiseEvent ();
	else
	{
		int i;
		for (i=0;i<brojPiva; i++)
			notify(0);
		Scheduler::Instance()->removeEvents(this);
	}
}

void Posetilac::notify (int ID)
{
	// Kada Scheduler obavesti Posetioca
	// on odlazi do Sanitarnog cvora
	// gde se izvrsava izbacivanje
	// jednog piva
	if (target)
	{
```

```
            send();
            brojPiva--;
      }
}
```

Treba da se u `Scheduler` doda još jedna funkcija da bi se ovo podržalo.

```
void removeEvents(ITimedElement *target);
```

Definicija funkcije izgleda ovako:

```
#include "Sched.h"

void Scheduler::removeEvents(ITimedElement *target)
{
      // Ponovo napuniti listu dogadjaja
      // sa svim vec postojecim dogadjajima
      // osim sa onima koji se odnose na target
      if (target == 0) return;

      // Pamti se pocetak postojece liste
      // i pocinje sa uporedjivanjem od pocetka
      Event* frs = first;
      Event *tmp = first;

      // Pocinje se nova lista
      first = 0;
      while (tmp!=0)
      {
            // Ako dogadjaj ne treba izbaciti
            if (tmp->getTarget() != target)
            {
                  // Dodaje se u novu listu
                  put(tmp);
            }
            // Prelazak na sledeci
            tmp = tmp->getNext();
      }
      // Nakon pravljenja nove liste (tj. izbacivanja nezeljenih)
      // mora se obrisati stara lista
      tmp = frs;

      while (frs!=0)
      {
            tmp = frs->getNext();
            delete frs;
            frs = tmp;
      }
}
```

d)

SanitarniCvor.h

```
#ifndef SanitarniCvor_h
#define SanitarniCvor_h 1

#include "FlowSink.h"
#include "Visitor.h"

// Klasa SanitarniCvor nasledjuje FlowSink
// jer je krajnja destinacija piva
class SanitarniCvor : public FlowSink
{
public:
      virtual void accept (Visitor* vis);
};
```

```
void SanitarniCvor::accept (Visitor* vis)
{
      if (vis) vis-> visitSanitarniCvor (this);
}

#endif
```

U klasu Visitor potrebno je dodati sledeće metode:

```
virtual void visitSanker(Sanker*);
virtual void visitPosetilac(Posetilac*);
virtual void visitSanitarniCvor(SanitarniCvor*);
```

Metode su realizovane, isto kao i ostale u klasi Visitor, kao *hook* metode:

```
inline void Visitor::visitSanker(Sanker*)
{
}

inline void Visitor::visitPosetilac(Posetilac*)
{
}

inline void Visitor::visitSanitarniCvor(SanitarniCvor*)
{
}
```