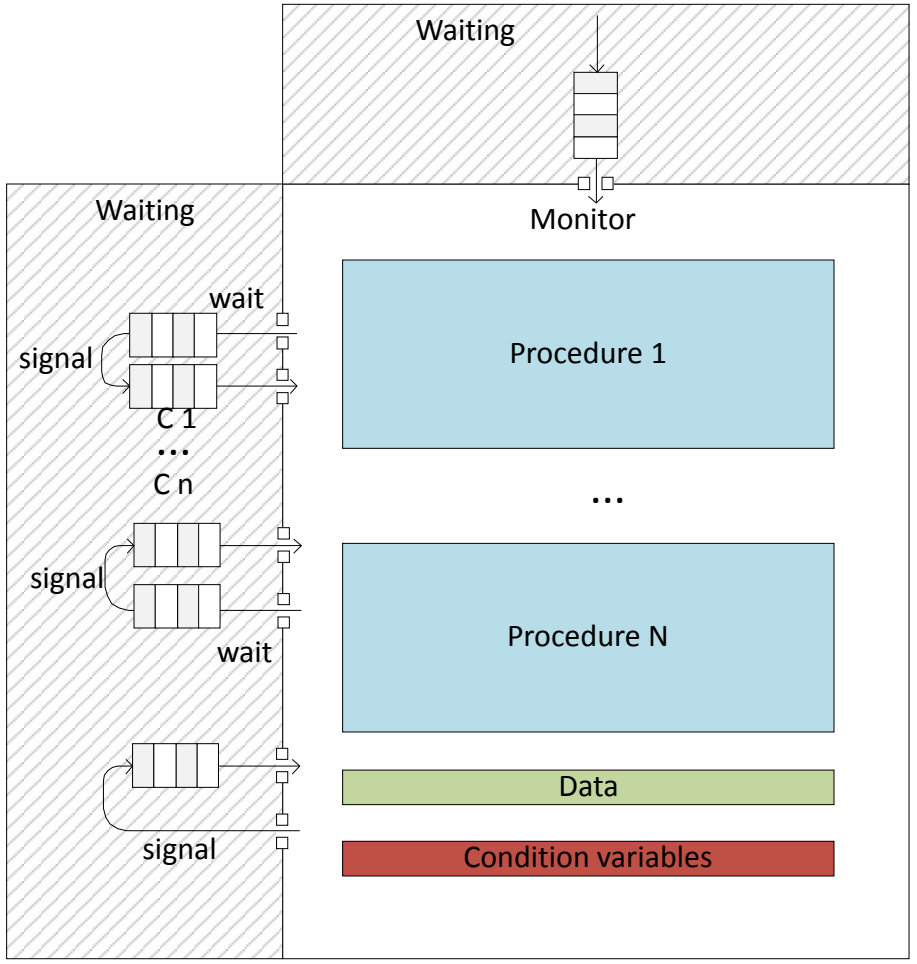
Operativni sistemi 2

Vežbe 2
Komunikacija i sinhronizacija

Lista predmeta:
IR smer - 13e113os2@lists.etf.rs
SI Smer - 13s113os2@lists.etf.rs

Monitori


Waiting
Waiting
Monitor
wait
signal
C 1
...
C n
Procedure 1
...
signal
wait
Procedure N
Data
signal
Condition variables

Zadatak

Implementirati monitor koji predstavlja ograničeni bafer koristeći semafore.

```
class Monitor {
public:
  Monitor () : mutex(1), itemAvailable (0),
spaceAvailable(0) {}
  int take ();
  void put (int x);
private:
  Semaphore mutex;

  Semaphore itemAvailable, spaceAvailable;
    // Condition vs. Semaphore
};
```

```
int Buffer::take () {
  mutex.wait();
    if (numberInBuffer == 0) {
      mutex.signal();
      itemAvailable.wait();
      mutex.wait();
    }


  //... remove the element
  tmp = ....
  numberInBuffer--;
  spaceAvailable.signal();

  mutex.signal();

  return tmp;
}
```

while

```
void Buffer::put (int x) {
  mutex.wait();

  ....

  numberInBuffer++;

  itemAvailable.signal();

  mutex.signal();
}
```

Druga varijanta

```
class Monitor {
public:
  Monitor () : mutex(1), itemAvailable (0),
spaceAvailable(0) {}
  int take ();
  void put (int x);
private:
  Semaphore mutex;

  Semaphore itemAvailable, spaceAvailable;
  int itemAvailable_c, spaceAvailable_c;
};
```

```
int Buffer::take () {
  mutex.wait();

    while (numberInBuffer == 0) {
      //wait(itemAvailable);
      itemAvailable_c++;
      mutex.signal();
      itemAvailable.wait();
      mutex.wait();
    }


  //... remove the element
  tmp = ....
  numberInBuffer--;

   //signal(spaceAvailable);
    if (spaceAvailable_c > 0){
      spaceAvailable.signal();
      spaceAvailable_c--;
    }

    mutex.signal();

    return tmp;
}
```

Problemi

1. Kako rešiti slučaj izlaska iz sredine operacije (sa return ili u slučaju izuzetka)?

```
int Monitor::criticalSection () {
  mutex.wait();
  return f()+2/x;  //  gde pozvati signal()?
}          //  šta ako dođe do izuzetka?
```

Rešenje:

```
 class Mutex {
public:
  Mutex (Semaphore* s) : sem(s){ if (sem) sem->wait(); }
 ~Mutex ()                     { if (sem) sem->signal(); }
private:
  Semaphore *sem;
};

void Monitor::criticalSection () {
  Mutex dummy(&sem);
  //... telo kritične sekcije
}
```

Problemi

2. Operacija oslobađanja kritične sekcije (`mutex.signal()`) i blokiranja na semaforu za čekanje na element (`itemAvailable.wait()`) moraju da budu nedeljive inače bi moglo da se dogodi da između ove dve operacije neki proces promeni uslov, a prvi proces se blokira na semaforu `itemAvailable` bez razloga.

Rešenje:

```
signalWait(mutex,itemAvailable); // atomicno - signalizira
                                 // na jednom semaforu,
                                // a blokira se na drugom
```

Zadatak - Februar 2008

• Procesi `Alarm` i `Passengers` prikazani dole treba da se sinhronizuju na sledeći način. Proces `Passengers` signalizira ulazak i izlazak putnika iz neke obezbeđene zone. Proces `Alarm` treba da čeka blokiran sve dok broj putnika koji su trenutno u obezbeđenoj zoni ne pređe neki prag `THRESHOLD`. Tada treba da se deblokira i uključi alarm pozivom operacije `alert`. Realizovati monitor `Signaller` koji obezbeđuje opisanu sinhronizaciju pomoću klasičnih uslovnih promenljivih.

```
process Alarm;
begin
  loop
    Signaller.waitForMax();
    alert;
  end;
end;
```

```
process Passengers;
begin
      loop
            Signaller.entry();
            Signaller.exit();
      end;
end;
```

```
monitor Signaller;
  export waitForMax, entry, exit;
  var
    count : integer;
    threshold : condition;
  procedure waitForMax ();
  begin
    while (count<=THRESHOLD) wait(threshold);
  end;

  procedure entry ();
  begin
    count := count + 1;
    if (count>THRESHOLD) signal(threshold);
  end;

  procedure exit ();
  begin
    count := count - 1;
  end;

Begin
  count := 0;
end;
```

Zadatak - Monitori u Javi

Na programskom jeziku Java implementirati ograničeni kružni bafer.
Napomena: Nit koja poziva `notify()` ili `notifyall()` nastavlja sa izvršavanjem u okviru monitora, dok probuđena(e) nit(i) čeka(ju) da prethodna nit oslobodi monitor i nakon toga zajedno sa svim ostalim nitima konkuriše(u) za ulazak u monitor.

```
public class Buffer{
 int n, kap;

 public synchronized void put(Object obj){
  while (n == kap) {
   try {
    wait();  // this.wait();
   } catch (InterruptedException e) { }
  }

  ++n;
  // store object

  notifyAll();
 }
```

```
public synchronized Object get(){
  while (n == 0) {
   try {
    wait();
   } catch (InterruptedException e) { }
  }


  --n;

  Object tmp = // get object

  notifyAll();

  return tmp;
 }

}
```

Bufer veličine 1

Nit1 -> get() -> blokira se na wait()
Nit2 -> get() -> blokira se na wait()
Nit3 -> put() -> odradi posao -> pozove notify()
Nit4 -> put() -> blokira se na wait()
Nit1 -> odblokira se -> odradi posao -> pozove notify()
Nit2 -> odblokira se -> blokira se na wait()

Nit2 i Nit4 blokirane (jedna proizvodjač, druga potrosač)!?

Zadatak – Oktobar 2015

Klasa `Server`, čiji je interfejs dat dole, služi kao jednoelementni bafer za razmenu podataka između proizvoljno mnogo uporednih niti proizvođača koji pozivaju operaciju `put` i potrošača koji pozivaju operaciju `get`. Tek kada jedan proizvođač upiše podatak tipa `Data` operacijom `put`, jedan potrošač može da ga pročita operacijom `get`; tek nakon toga neki proizvođač sme da upiše novi podatak, i tako dalje naizmenično. Na jeziku Java implementirati klasu `Server` sa potrebnom sinhronizacijom.

```
class Server {
        public void put (Data d);
        public Data get ();
}
```

```
class Server {
  private Data d;
  private bool readyToRead = false,
               readyToWrite = true;
  public synchronized void put (Data data) {
    while (!readyToWrite)wait();
      readyToWrite=false;
      d=data; readyToRead=true;
      notifyAll();
  }
  public synhronized Data get () {
    while (!readyToRead) wait();
    readyToRead=false;
    Data data=d;
    readyToWrite=true;
    notifyAll();
    return data;
  }
}
```

Zadatak – Septembar 2014

Korišćenjem klasičnih uslovnih promenljivih, napisati kod za monitor koji ima dve operacije, *flip* i *flop*, uz sledeću sinhronizaciju: između dva susedna izvršavanja operacije *flip* mora se najmanje *N* puta izvršiti operacija *flop*. Drugim rečima, nakon jednog izvršavanja *flip*, mora doći najmanje *N* (može i više) izvršavanja *flop*, pa onda opet može *flip* itd.

```
monitor FlipFlop;
export flip, flop;
var i : integer;
    cond : condition;
procedure flip ();
begin
  while i<N do
    cond.wait;
  i:=0;
 (* do flip *)
end;
procedure flop ();
begin
  (* do flop *)
  if i<N then
    i:=i+1;
  cond.signal;
end;
begin
  i:=N;
end; (* monitor *)
```

Zadatak – Oktobar 2012

Jedna varijanta uslovne sinhronizacije unutar monitora je sledeća. Svaki monitor ima samo jednu, implicitno definisanu i anonimnu (bez imena) uslovnu promenljivu, tako da se u monitoru mogu pozivati sledeće dve sinhronizacione operacije:

wait(): bezuslovno blokira pozivajući proces i oslobađa ulaz u monitor;

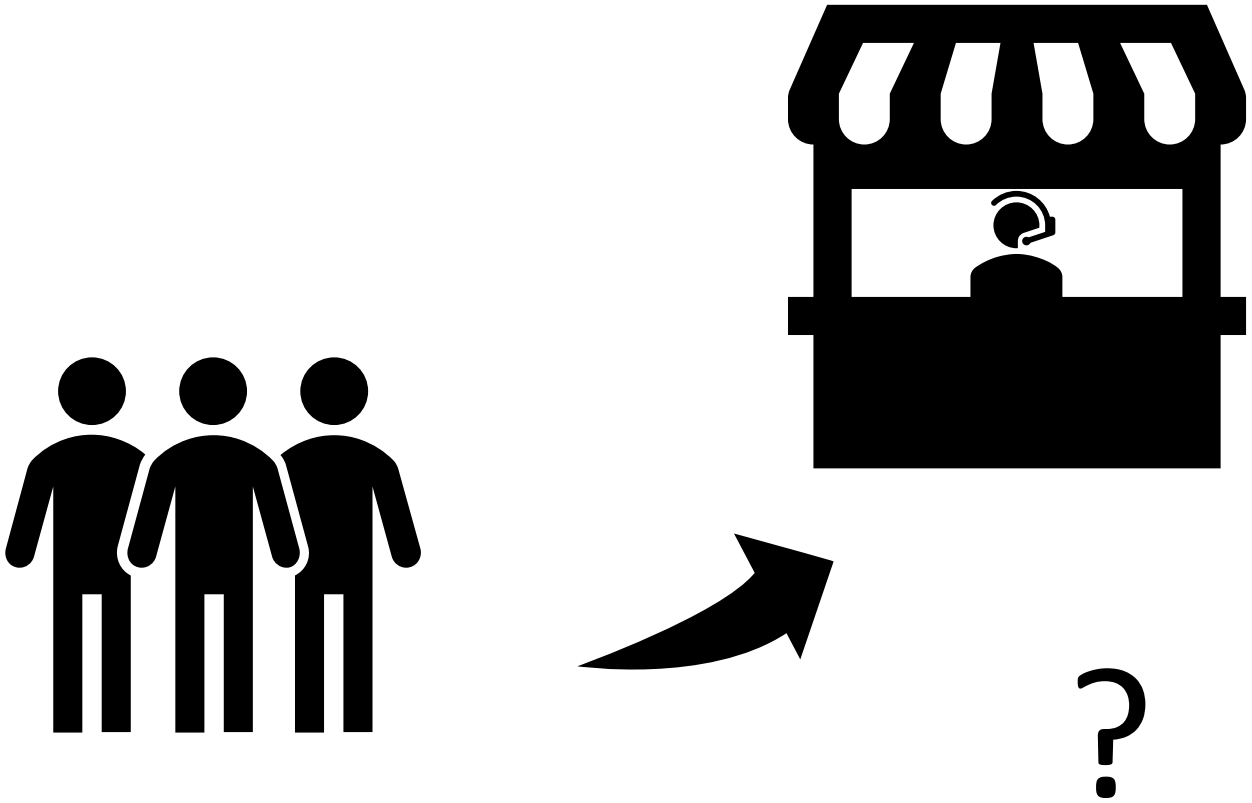
notifyAll(): deblokira *sve* procese koji čekaju na uslovnoj promenljivoj, ako takvih ima (naravno, mešusobno isključenje tih procesa je i dalje obezbeđeno).

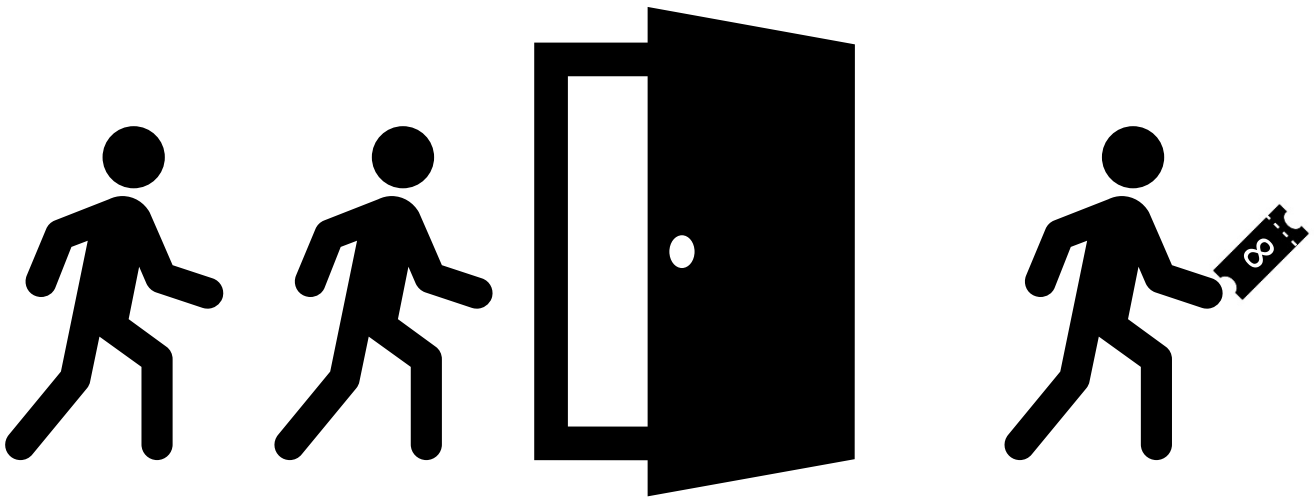
Projektuje se konkurentni klijent-server sistem. Server treba modelovati monitorom sa opisanom uslovnom promenljivom. Klijenti su procesi koji ciklično obavljaju svoje aktivnosti. Pre nego što u jednom ciklusu neki klijent započne svoju aktivnost, dužan je da od servera traži dozvolu u obliku "žetona" (*token*). Kada dobije žeton, klijent započinje aktivnost. Po završetku aktivnosti, klijent vraća žeton serveru. Server vodi računa da u jednom trenutku ne može biti izdato više od N žetona: ukoliko klijent traži žeton, a ne može da ga dobije jer je nema žetona, klijent se blokira. Napisati kod monitora i procesa-klijenta.

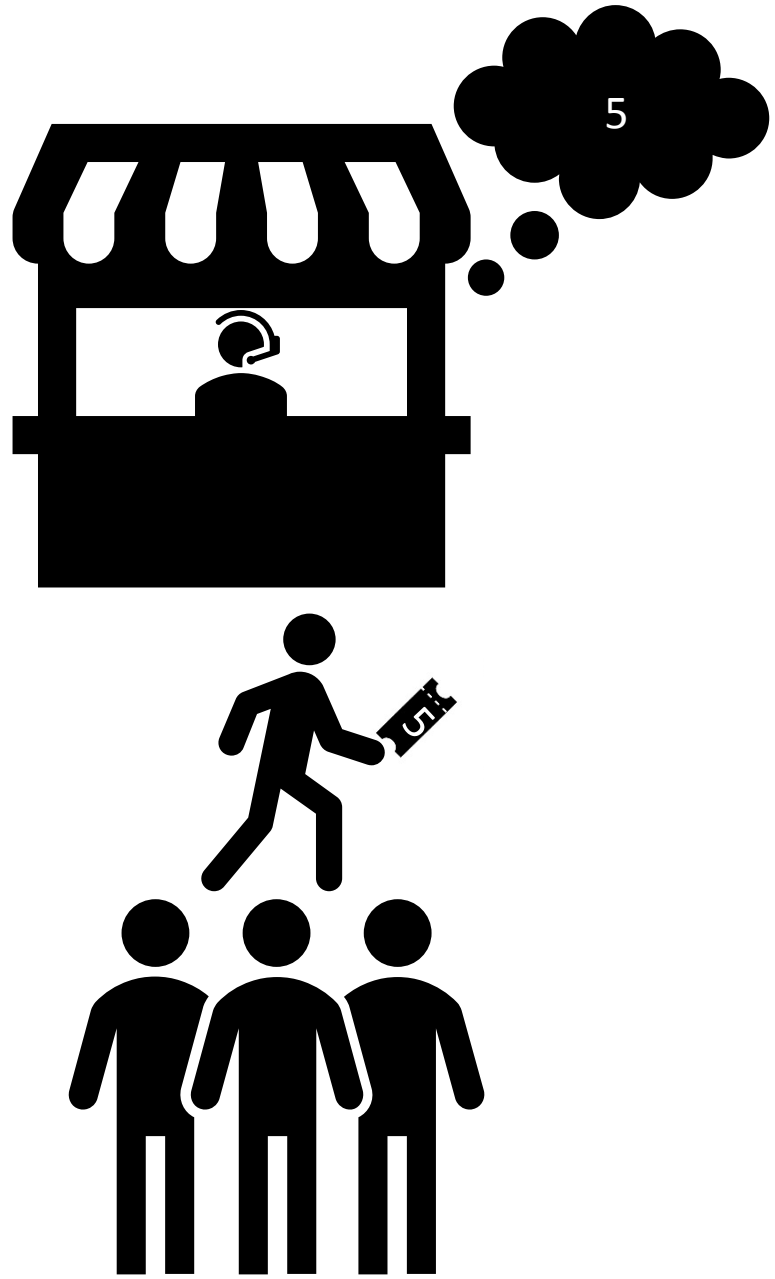
```
monitor server;
        export acquireToken, returnToken;
        var numOfTokens : integer;

procedure acquireToken (); begin
        while numOfTokens<=0 do wait();
         numOfTokens := numOfTokens - 1;
end;

procedure returnToken (); begin
        numOfTokens := numOfTokens + 1;
        notifyAll();
end;
begin
        numOfTokens := N;
end; (* server *)
task type client; begin
loop
        server.acquireToken; do_some_activity;
        server.returnToken;
end;

end; (* client *)
```

Zadatak – Februar 2010

• Na programskom jeziku Java, korišćenjem sinhornizovanih metoda (*synchronized*), implementirati ograničeni kružni bafer veličine N celobrojnih podataka tipa int. Pri umetanju podataka u bafer, umeće se po K podataka, gde je K konstanta koja se definiše pri kreiranju bafera. Podaci se u bafer umeću atomično. Ukoliko u baferu nema dovoljno mesta, pozivajuća nit se blokira, ali tako da se obezbedi fer pristup. To znači da nitima treba obezbediti onaj redosled umetanja elemenata u bafer u kojem su prvi put pokušale da umetnu niz od K podataka. Pri dohvatanju podataka iz bafera, dohvata se jedan po jedan podatak. Ukoliko nema dovoljno podataka, pozivajuću nit blokirati. U slučaju dohvatanja podataka nije potrebno voditi računa o fer redosledu dohvatanja podataka. Zadatak rešiti uz ograničenje da jedine sinhronizovane metode mogu biti metode klase koja predstavlja implementirani bafer. Poznato je da je K<N, ali nije poznato da li je N deljivo sa K.




8


5
5

```
public class buffer {
  private int data[];
  private int K;
  private int N;
  private int free;   //broj slobodnih mesta u baferu
  private int tail;   //pozicija sa koje se uzima sledeci
podatak
  private int head;   //pozicija na koju se smesta sledeci
podatak
  private int tiket;   //uzimanje rednog broja
  private int sledeci; //redni broj koji je na redu za
umetanje u bafer
                       //kada se pojavi dovoljno slobodnog
mesta
    public buffer(int N, int K) {
        data = new int[N];
        this.N = N;
        free = N;
        this.K = K;
        head = 0;
        tail = 0;
        tiket = 0;
        sledeci = 0;
    }
```

```
public synchronized void put(int d[]) throws
InterruptedException{
        int red;
        if (free<K || sledeci != tiket) {
            red = tiket++;
            while(free<K || sledeci!=red){
                wait();
            }
            sledeci++;
        }
        for(int i =0; i<K; i++){
            data[head]=d[i];
            head = (head+1)%N;
        }
        free -= K;
        notifyAll();
    }
```

```
 public synchronized int get() throws
InterruptedException{
        while (free == N){
            wait();
        }
        int ret = data[tail];
        tail = (tail+1)%N;
        free++;
        notifyAll();
        return ret;
}
```

Zadatak – Decembar 2014

Korišćenjem klasičinih monitora i uslovnih promenljivih, realizovati monitor `TaxiDispatcher` koji implementira sledeće ponašanje dispečera taksija. Korisnik (*user*) i taksi vozilo (*taxi*) su procesi koji se prijavljuju dispečeru pozivom procedura `userRequest` i `taxiAvailable`, respektivno.

Korisnik poziva proceduru `userRequest` kada želi da dobije vožnju taksijem. Ako trenutno ima raspoloživih taksi vozila koja čekaju u redu na zahtev korisnika, korisniku će odmah biti dodeljeno jedno od tih raspoloživih vozila. U suprotnom, korisnik će biti blokiran i čekaće u redu korisnika sve dok mu dispečer ne dodeli vozilo. Broj dodeljenog taksi vozila korisnik dobija u izlaznom parametru `assignedTaxiID`.

Taksi vozač poziva proceduru `taxiAvailable` kada je slobodan da primi i preveze korisnika. Kao ulazni parametar `myID` dostavlja svoj broj vozila. Ako trenutno ima korisnika koji čekaju u redu na raspoloživo vozilo, tom vozilu će biti dodeljen jedan od tih korisnika. U suprotnom, taksi će biti blokiran i čekaće u redu raspoloživih vozila sve dok mu dispečer ne dodeli korisnika.

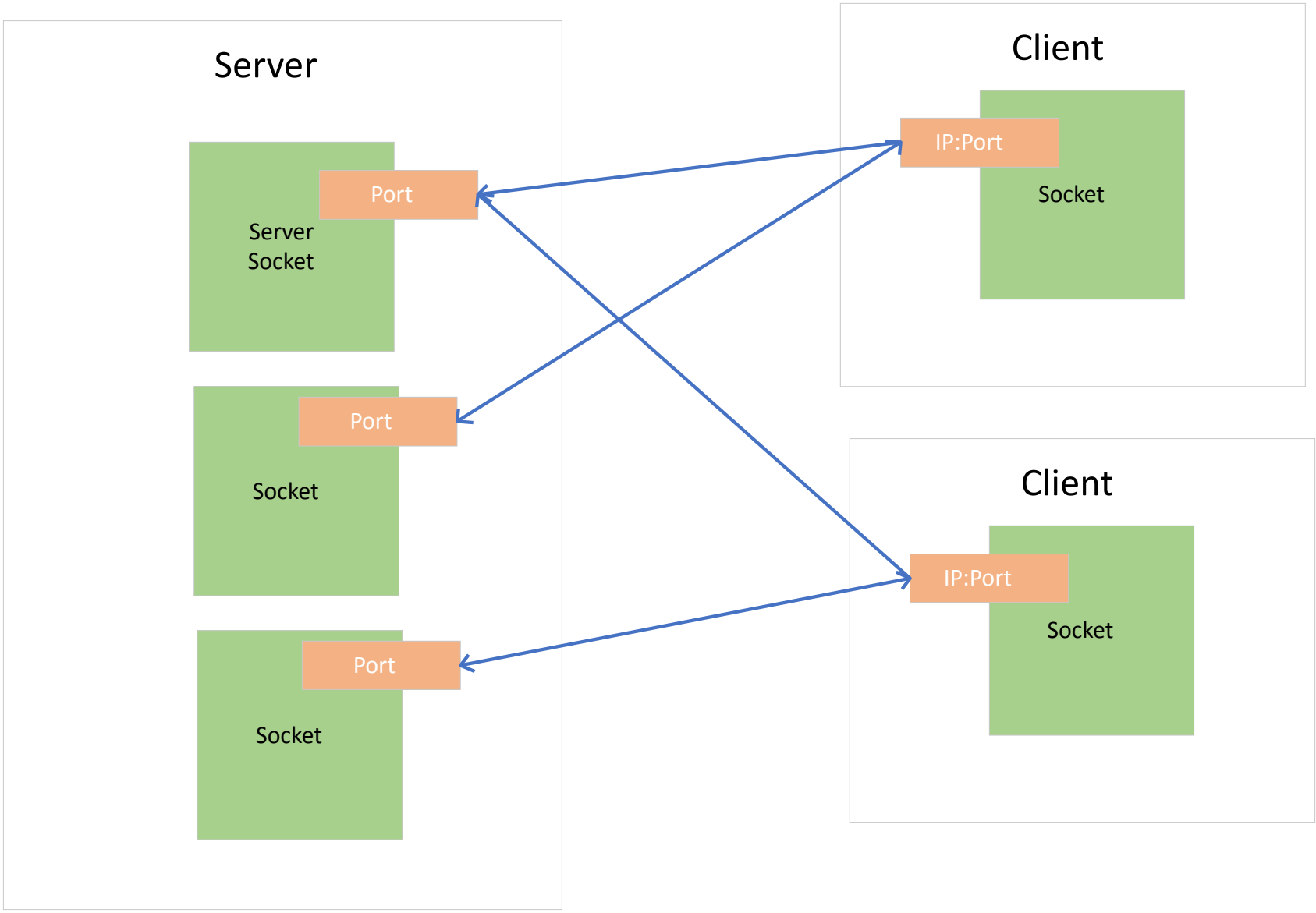
Semantika uslovnih promenljivih je takva da proces koji je čekao na uslovnoj promenljivoj i oslobođen je ima prioritet u nastavku izvršavanja u odnosu na proces koji je signalizirao tu uslovnu promenljivu. Oba ta procesa, naravno, imaju prioritet u odnosu na druge procese koji čekaju na ulaz u monitor.

```
monitor TaxiDispatcher;
export userRequest, taxiAvailable;

var waitingUsers, availableTaxis : integer;
waitForUser, waitForTaxi : condition;
taxiID : integer;

procedure userRequest (var assignedTaxiID : integer);
begin
  if (availableTaxis>0) begin
   availableTaxis:=availableTaxis-1; waitForUser.signal;
   assignedTaxiID:=taxiID;
  end else
 begin
   waitingUsers:=waitingUsers+1; waitForTaxi.wait;
   assignedTaxiID:=taxiID;
 end;
end;
```

```
procedure taxiAvailable (myID : integer); begin
    if (waitingUsers>0) begin
     waitingUsers:=waitingUsers-1; taxiID:=myID;
     waitForTaxi.signal;
    end
    else
   begin
     availableTaxis:=availableTaxis+1; waitForUser.wait;
      taxiID:=myID;
      end;
end;

begin
  waitingUsers:=0; availableTaxis:=0;
end;
```

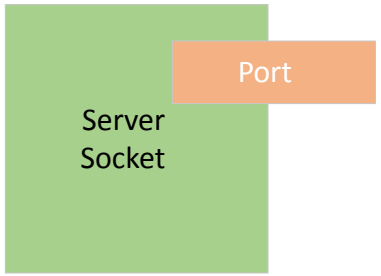
Java Sockets

- Tutoriali:
- http://docs.oracle.com/javase/tutorial/networking/sockets/index.html
- http://www.oracle.com/technetwork/java/socket-140484.html

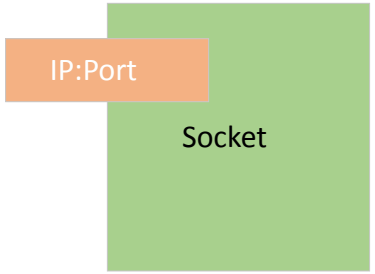

Server
Port
Server
Socket
Port
Socket
Port
Socket
Client
IP:Port
Socket
Client
IP:Port
Socket

API

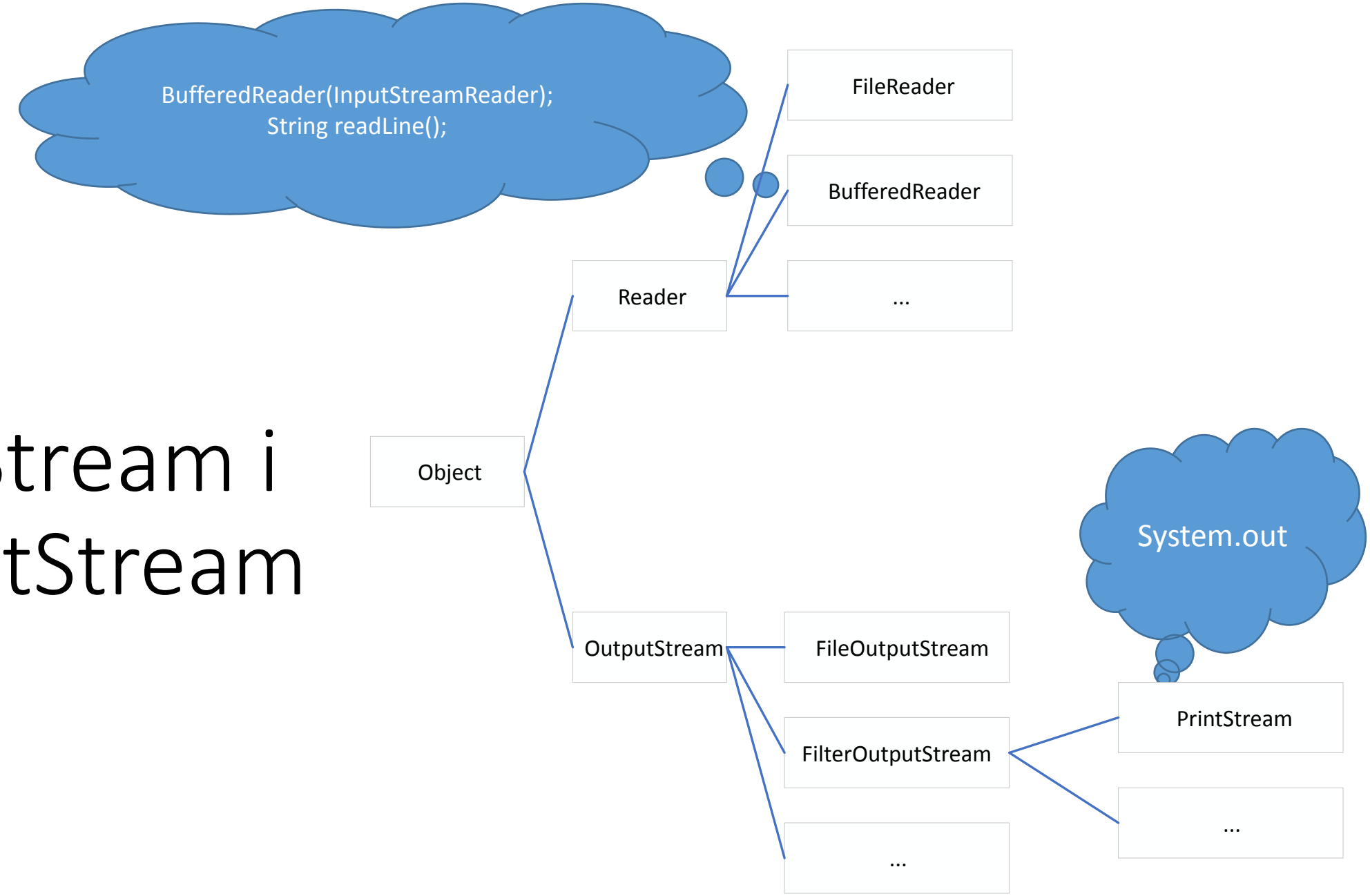
ServerSocket(int port);
Socket accept() throws IOException;



Socket(String host, int port) throws UnknownHostException, IOException;
InputStream getInputStream() throws IOException;
OutputStream getOutputStream() throws IOException



InputStream i OutputStream



Zadatak - Školski Web service u Javi

Data je klasa ServerService:

```
public class ServerService {
     protected int i = 0;
     public int setI(int ni) { int tmp = i; i = ni;
   return tmp;   }
     public int add(int a, int b) { return a+b+i; }
} // End ServerService.java
```

Implementirati potrebne klase koje će krajnem korisniku obezbediti identičan interfejs kao i klasa ServerService i koje će mu omogućiti da sa udaljenog računara kroz obezbeđeni interfejs poziva operacije klase ServerService.

Zadatak – Jun 2006

Implementirati veb servis (*Web Service*), sa svim potrebnim delovima i na klijentskoj i na serverskoj strani, na programskom jeziku Java, koji će krajnjem korisniku pružiti sledeći interfejs:

```
public class Service{
        public Data calc(Data d){…}
        public double calc(Data d1, Data d2) {…}
}
```

Metodi `Service.calc` vrše odgovarajuće obrade nad podacima tipa `Data`, a zatim vraćaju odgovarajući rezultat. Korisnik treba da instancira objekat klase `Service` na svojoj, klijenstkoj strani, koji će predstavljati posrednik (*proxy*) do stvarnog objekta na serverskoj strani koji vrši stvarno izračunavanje. Međuprocesnu komunikaciju realizovati preko priključnica (*socket*) i razmenom poruka (*message passing*). Dozvoljeno je korišćenje koda prikazanog na vežbama. Interfejs klase `Data` dat je sledećim kodom:

```
public Data{
        public Data(String str){…}
        public String serialize(){…}
        …
}
```

Metod `Data.serialize` služi da napravi string reprezentaciju odgovarajućeg objekta, iz koga se može izvršiti rekonstrukcija tog objekta pomoću konstruktora `Data(String)`. Smatrati da ovaj string neće imati znak `'#'` u sebi.

Slično prethodnom zadatku, uz sledeće izmene: kada je potrebno poslati podatak tipa `Data`, treba poslati string: d.serialize(). Kada je potrebno primiti podatak tipa Data, onda treba izvršiti konverziju primljene poruke (nastale sa d.serialize()) u objekat tipa Data pomoću postojećeg konstruktora:

```
message = ...;
return new Data(message);
```

Podatke tipa double, slati i primati analogno slanju i prijemu podataka tipa int.

Zadatak – April 2006

Date su klase `Base` i `Derived` na programskom jeziku Java.

```
public class Base {
	public int f(int a, int b) { ... }
	public int f(int a) { ... }
}

public class Derived extends Base {
	public int f(int a, int b) { ... }
}
```

Klasa `Base` ima dve operacije `f` sa preklopljenim imenom (engl. *overloaded*). Klasa `Derived` je izvedena iz klase `Base` i u njoj je redefinisan metod `f(int,int)` (engl. *overrided*). Implementirati klase `BaseProxy` i `DerivedProxy`, koje će se instancirati na klijentskoj strani, a koje će predstavljati posrednike (engl. *proxy*), do stvarnih objekata na serverskoj strani koji izvršavaju stvarno izračunavanje. Međuprocesnu komunikaciju realizovati preko priključnica (engl. *socket*) i razmenom poruka (engl. *message passing*). Dozvoljeno je korišćenje koda prikazanog na vežbama.

Jedina bitna razlika u odnosu na prethodne zadatke je da u poruku pored naziva operacije mora biti uključen i naziv tipa, kako bi se razlikovale metode iz dve prikazane klase. Takođe, u zavisnosti od broja prosleđenih parametara treba voditi računa koja se operacija klase `Base` poziva.

Zadatak – Oktobar 2006

Implementirati Web Service, na programskom jeziku Java, koji će krajnjem korisniku pružiti sledeći interfejs:

```
public class NetSemaphore {
      public NetSemaphore(String host, int port){…}
      public void create(String name, int initialValue){…}
      public void signalS(String name){…}
      public void waitS(String name){…}
}
```

Servis treba da obezbedi sinhronizaciju niti koje se izvršavaju na udaljenim računarima. `NetSemaphore` se povezuje sa odgovarajućim serverom preko koga se vrši sinhronizacija. Metode `create`, `signalS` i `waitS` imaju semantiku primitiva za rad sa standardnim brojačkim semaforima, a služe da kreiraju, signaliziraju i čekaju na semafor `name` koji se nalazi na serveru. Ako je semafor sa imenom `name` već postoji, onda primitiva `create` nema efekta, već se koristi postojeći semafor. Ako se pozivaju primitive `signalS` i `waitS` na semaforima koji ne postoje, ignorisati te pozive. Pretpostaviti da postoji klasa `Semaphore` koja implementira standardni brojački semafor i ima sledeći interfejs:

```
public class Semaphore {
    public Semaphore(int initialValue){…}
    public void signalS(){…}
    public void waitS(){…}
}
```

Međuprocesnu komunikaciju realizovati preko priključnica (*socket*) i razmenom poruka (*message passing*). Dozvoljeno je korišćenje koda prikazanog na vežbama (kod sa vežbi ne treba prepisivati, nego npr. reći koja klasa ili koji metod se koriste i/ili menjaju, nasleđuju, ...).

```
public class NetSemaphore extends Service {
  public NetSemaphore(String host, int port){
    super(host, port);
  }
  public void waitS(String name) {
    String message = "#wait#" + name + "#";
    sendMessage(message);
    receiveMessage();
  }
 …
}
```

U Server dodati:

```
private Map<String, Integer> semaphores = new HashMap<String, Integer>();
    public synchronized void createSemphore(String name, int init) {
        semaphores.put(name, init);
    }
    public synchronized void waitS(String name) {
        if (semaphores.containsKey(name)) {
            int value = semaphores.get(name);
            while (value < 1) {
                try {
                    wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            value--;
            semaphores.put(name, value);
        }
    }
    public synchronized void signalS(String name) {
        if (semaphores.containsKey(name)) {
            int value = semaphores.get(name);
            value++;
            semaphores.put(name, value);
            notify();
        }
    }
```

U RequestHandler dodati:

```
private final Server server;
   public RequestHandler(Service service, Server server) {
       this.server = server;
     ...
```

Operacija wait:

```
server.waitS(name);
```

Zadatak – oktobar 2011

Na jeziku Java implementirati serverski proces koji predstavlja agenta na aukciji. Ovaj proces treba da „osluškuje“ port 1025 preko koga prima poruku za otvaranje nadmetanja sa početnom cenom. Zatim, nakon zatvaranja prethodne konekcije, po istom portu počinje da prihvata ponude od ostalih učesnika u nadmetanju, pri čemu pamti trenutno najveću ponudu. U svakoj ponudi učesnik se identifikuje svojom vrednošću priključnice (IP adresa i port računara preko koga prima odgovor). Svaka nova ponuda mora biti veća od prethodne, inače se ponuđaču odmah vraća informacija o odbijanju ponude. U suprotnom se vraća poruka da je ponuda prihvaćena i da se čeka krajnji ishod nadmetanja. U slučaju da u međuvremenu pristigne ponuda sa većom vrednošću, vraća se informacija o odbijanju ponude, a ukoliko među prethodnih 5 ponuda ne stigne ni jedna veća, nadmetanje se zatvara, a procesu koji predstavlja učesnika sa najvišom ponudom šalje se poruka o pobedi. Za sinhronizaciju i komunikaciju koristiti priključnice (*sockets*) i mehanizam prosleđivanja poruka (*message passing*).

```
public class Agent {

static int bestOffer;
static String bestClientHost = null;
static int bestClientPort;
static boolean auctionOver = false;
static int badOfferCounter = 0;
static BufferedReader in;
static PrintWriter out;

public static void main(String[] args) {
 try {
  ServerSocket sock = new ServerSocket(1025);
  Socket clientSocket = sock.accept();
  in = new BufferedReader(new InputStreamReader(
    clientSocket.getInputStream()));
  StringTokenizer st = new StringTokenizer(in.readLine(), "#");
  if (!st.nextToken().equals("StartAuction")) {
    auctionOver = true;
  }else {
    bestOffer = Integer.parseInt(st.nextToken());}

  clientSocket.close();

  while (!auctionOver) {
   clientSocket = sock.accept();
```

```
in = new BufferedReader(new
          InputStreamReader(clientSocket.getInputStream()));
out = new PrintWriter(clientSocket.getOutputStream(), true);
st = new StringTokenizer(in.readLine(), "#");
String clientHost = st.nextToken();
String clientPort = st.nextToken();
int newOffer = Integer.parseInt(st.nextToken());

if (newOffer > bestOffer) {
 if (bestClientHost != null)
   sendMsgToBestClient("BetterOfferReceived");
 bestOffer = newOffer;
 bestClientHost = clientHost;
 bestClientPort = Integer.parseInt(clientPort);
 out.println("OfferAccepted");
 badOfferCounter = 0;
} else {
 out.println("OfferRejected");
 badOfferCounter++;
}
clientSocket.close();
```

```
   if (badOfferCounter == 5) {
    if (bestClientHost != null) sendMsgToBestClient("YouWon!");
    auctionOver = true;
   }
  }
 } catch (Exception e) { System.err.println(e);}
}

static void sendMsgToBestClient(String msg) throws
                              UnknownHostException, IOException {
 Socket clientSocket = new Socket(bestClientHost,bestClientPort);
 PrintWriter oldOut = new PrintWriter(clientSocket.getOutputStream(),true);
 oldOut.println(msg);
 clientSocket.close();
 }
}
```