# Mrežno računarstvo

poglavlje 13.

UDP Datagrams and Sockets

# Uvodna priča

- **U prethodnim poglavljima razmatrane su aplikacije koje koristeTCP protokol.**
- TCP je dizajniran za pouzdan transfer podataka. Ako se prilikom transfera podaci oštete ili izgube, TCP obezbeđuje da se ponovo pošalju, ako ne stižu odgovarajućim redom, TCP ih preuredi, ako podaci stižu prebrzo, TCP uspori tako da se paketi ne gube.
- Međutim, cena ove pouzdanosti je brzina. **Uspostavljanje i raskidanje konekcije može trajati prilično vremena, posebno za protokole kao što je HTTP, koji zahteva puno kratkih transfera**

# UDP

- User Datagram Protocol (UDP) je alternativni protokol za slanje podataka preko IP, koji je vrlo brz, ali nije pouzdan
- Kada se pošalju podaci, nema načina da se sazna da li su stigli ili da li su stigli redosledom kojim su poslati. Međutim, podaci koji stižu, stižu brzo

# UDP protokol

- Zašto bi iko želeo da koristi nepouzdani protokol?
- **UDP očigledno nije dobar za aplikacije poput FTP.**
- Međutim, postoje mnoge vrste aplikacija u kojima je brzina važnija od ispravnog primanja svakog pojedinačnog bita.
- real-time audio ili video

- U drugim aplikacijama, mogu se implementirati testovi pouzdanosti u application sloju. Npr. ako klijent pošalje kratak UDP zahtev serveru, može pretpostaviti da je paket izgubljen ako ne dobije odgovor u dogovorenom periodu. To je jedan način na koji funkcioniše Domain Name Server (DNS). **DNS takođe može raditi i preko TCP.**
- ...

- **Razlika između TCP i UDP često se objašnjava analogijom između telefonskog i poštanskog sistema.**
- TCP je poput telefonskog sistema. Kada pozovete broj, telefon se javi i konekcija između dve strane je uspostavljena. Dok pričate, znate da druga strana čuje vaše reči onim redom kojim ih pričate. Ako je telefon zauzet ili se niko ne javlja, odmah to znate.
- UDP, suprotno, je kao poštanski sistem. Šaljete poštu na neku adresu. Većina pisama stigne, ali neka mogu biti izgubljena. Pisma verovatno stižu redom kojim su poslata, ali ne postoji garancija. Što smo dalje od primaoca, verovatnije je da će pošta biti izgubljena ili stići nekim drugim redosledom. Ako je to problem, možemo pisati redne brojeve, a zatim tražiti od primalaca da ih urede i pošalju pismo koje kaže koja pisma su stigla tako da možemo ponovo da pošaljemo ona koja nisu. Međutim, mi i primalac to moramo da dogovorimo unapred. Pošta to neće uraditi za nas.

- I telefonski i poštanski sistem imaju svoje korisnike. Iako se bilo koji može koristiti za skoro svaku komunikaciju, u nekim slučajevima jedan je definitivno superiorniji od drugog. Isto važi i za UDP i TCP.
- TCP aplikacije su uobičajenije od UDP.
- **Multicasting se zasniva na UDP: multicast soket je jednostavna varijanta UDP soketa**

- Java implementacija UDP-a podeljena je u dve klase: DatagramPacket i DatagramSocket.
- Klasa DatagramPacket puni bajtovima podataka UDP pakete koji se zovu datagram-ima i dopušta da ispraznimo datagram-e koje primimo.
- DatagramSocket šalje i prima UDP datagrame. Da bismo poslali podatke, stavimo ih u DatagramPacket i pošaljemo paket koristeći DatagramSocket. Da primimo podatke, primimo DatagramPacket objekat od DatagramSocket-a, a zatim pročitamo sadržaj paketa.

- Sami soketi su vrlo jednostavni.
- U UDP, sve o datagramu, uključujući adresu na koju je upućen, uključeno je u sam paket. Soket samo treba da zna lokalni port na kome sluša ili šalje
- Ova podela posla je u suprotnosti sa klasama Socket i ServerSocket koje koristi TCP.
- Prvo, UDP nema ideju o jedinstvenoj konekciji između hostova. Jedan soket šalje i prima sve podatke upućene ka ili od porta bez ikakve brige ko je udaljeni host. **Jedan DatagramSocket može slati podatke i primati podatke od mnogo nezavisnih hostova. Soket nije posvećen jednoj konekciji, kao što je to slučaj u TCP.** Zapravo, UDP nema nikakav koncept konekcije između dva hosta; on jedino poznaje pojedinačne datagrame. Otkrivanje ko šalje podatke je odgovornost aplikacije.

# Klasa DatagramPacket

- UDP datagrami dodaju jako malo IP datagramima od kojih nastaju. UDP zaglavlje dodaje svega 8 bajtova IP zaglavlju.
- UDP zaglavlje uključuje port brojeve izvorišta i odredišta, dužinu svega što prati IP zaglavlje i opcionu kontrolnu sumu
- Pošto se portovi zadaju kao 2-bajtni neoznačeni brojevi, 65536 različitih mogućih UDP portova je dostupno po hostu. Oni se razlikuju od 65536 različitih TCP portova po hostu.
- Dužina je takođe 2-bajtni neoznačeni integer, pa je broj bajtova u datagram-u ograničen na 65536-8 za zaglavlje. Međutim, to je teoretska granica. U praksi je veličina znatno manja
- Kontrolna suma je opciona i programi iz aplikacionog sloja je ne koriste niti joj mogu pristupiti. Ako se kontrolna suma ne poklopi, native mrežni software tiho odbaci datagram, ne obaveštavajući o tome ni pošiljaoca ni primaoca. Na kraju krajeva, UDP je nepouzdan protokol.

- **Količina podataka u UDP datagramu je na mnogim platformama ograničena na 8192 bajta (8K)**
- Implementacije ne moraju prihvatiti datagrame sa više od 576 bajtova ukupno, uključujući podatke i zaglavlja.
- Zato, treba biti jako obazriv sa programima koji šalju ili primaju UDP pakete sa više od 8K podataka. Veći deo vremena, veći paketi se jednostavno skrate na 8K. Za maksimalnu bezbednost, data deo UDP paketa treba da bude 512 bajtova ili manji, iako se to ograničenje negativno odražava na performanse u poređenju sa paketima veće veličine
- (ovo je problem i za TCP datagrame, ali stream-based API koji obezbeđuju Socket i ServerSocket potpuno štiti programere od ovih detalja)

- U Javi, UDP datagram je predstavljen instancom klase DatagramPacket
- Ova klasa obezbeđuje metode za dohvatanje i postavljanje izvorišne i odredišne adrese u IP zaglavlju, za dohvatanje i postavljanje izvorišnog i odredišnog porta, za dohvatanje i postavljanje podataka i dužine podataka
- Preostala polja zaglavlja nisu pristupačna Java kodu

# Konstruktori

- DatagramPacket koristi razne konstruktore u zavisnosti od toga da li će se paket koristiti za slanje podataka ili njihovo primanje **(što je malo neuobičajeno – korišćenje različitih konstruktora za kreiranje objekata koji će se koristiti u različitim kontekstima)**
- Svi konstruktori (njih 6) primaju kao argumente byte niz koji čuva podatke datagrama i broj bajtova u tom nizu koji se koristi za podatke datagrama. Kada želimo da primimo datagram, trebaju nam samo ti argumenti i još niz treba da bude prazan. Kada soket prima datagram iz mreže, podatke datagrama smešta u niz DatagramPacket objekta, do dužine koju smo zadali.

- Drugi skup konstruktora se koristi za kreiranje datagrama koje šaljemo preko mreže. Kao i prvi, ovi konstruktori zahtevaju bafer-niz i dužinu, ali takođe i InetAddress i port na koji se paket šalje. U ovom slučaju, konstruktoru se prosleđuje byte niz koji sadrži podatke koje želimo da pošaljemo, kao i odredišna adresa i port na koji se paket šalje. DatagramSocket čita odredišnu adresu i port iz paketa. Adresa i port se ne smeštaju u soket, kao u TCP.

# Konstruktori za primanje datagrama

- 2 konstruktora kreiraju novi DatagramPacket objekat za primanje  podataka iz mreže
- public DatagramPacket(byte[] buffer, int length)
- public DatagramPacket(byte[] buffer, int offset, int length)
- Kada soket primi datagram, smešta podatke datagrama u buffer počev od buffer[0] (prvi), odnosno buffer[offset] (drugi) sve dok se ceo paket ne pročita ili dok se ne upiše length bajtova u buffer.
- length mora biti manje ili jednako od buffer.length-offset
- IllegalArgumentException (RuntimeException – naš kod nije dužan da ga hvata)
- sledeći fragment koda kreira novi DatagramPacket za primanje do 8192 bajta:

```
byte[] buffer = new byte[8192];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
```

# 8K

- Konstruktor ne mari za veličinu bafera i dopustio bi da kreiramo DatagramPacket sa megabajtima podataka. Ali...(važi ona prethodna priča o ograničenjima) ne treba kreirati DatagramPacket objekte sa više od 8192 bajtova podataka

# Konstruktori za slanje datagrama

- 4 konstruktora kreiraju novi DatagramPacket objekat za slanje podataka preko mreže
- public DatagramPacket(byte[] data, int length, InetAddress destination, int port)
- public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port)
- public DatagramPacket(byte[] data, int length, SocketAddress destination)
- public DatagramPacket(byte[] data, int offset, int length, SocketAddress destination)

- Svaki od ovih konstruktora kreira novi DatagramPacket koji će biti poslat drugom hostu.
- Paket se puni sa length bajtova niza data počev od offset ili 0, ako se ne koristi offset.
- IllegalArgumentException (length>data.length)
- length bajtova podataka se šalje preko mreže

# Izbor veličine Datagram-a

- Količina podataka kojom se puni paket zavisi od situacije. Neki protokoli diktiraju veličinu paketa. Npr. rlogin odašilje svaki karakter udaljenom sistemu skoro odmah pošto ga korisnik otkuca. Tu su paketi kratki: jedan bajt podataka i nekoliko bajtova zaglavlja. Druge aplikacije nisu takve: npr. file transfer je mnogo efikasniji sa većim baferima; jedini zahtev je da podelimo fajl u pakete ne veće od maksimalne dozvoljene veličine paketa.
- Nekoliko faktora je uključeno u izbor optimalne veličine paketa. Ako je mreža jako nepouzdana, bolji su manji paketi pošto su manje šanse da budu oštećeni u putu. S druge strane, vrlo brzi i pouzdani LAN-ovi treba da koriste pakete što je moguće veće. 8K = 8192 bajta je dobar kompromis za mnoge vrste mreža

- Uobičajeno je konvertovati podatke u byte niz i smestiti ih u data pre kreiranja DatagramPacket, ali nije apsolutno neophodno.
- Promene niza data nakon kreiranja datagrama i pre njegovog slanja menjaju podatke u datagramu (podaci se ne kopiraju u privatni bafer). U nekim aplikacijama, može se iskoristiti prednost ovoga. Npr. možemo smestiti u data podatke koji se vremenom menjaju i slati tekući datagram (sa ”najtazijim” podacima) svakog minuta. Međutim, važnije je osigurati da se podaci ne menjaju kada to ne želimo. Ovo je posebno važno kada nam je program multithreaded i razne niti mogu pisati podatke u bafer data. Ako je to slučaj, vrši se sinhronizacija data promenljive ili kopiraju podaci u privremeni bafer pre konstruisanja DatagramPacket-a.

- Sledeći fragment kreira DatagramPacket napunjen podacima ”This is a test” u ASCII-ju. Paket je upućen portu 7 (echo port) na hostu ”www.ibiblio.org”:

```
String s = ”This is a test”;
byte[] data = s.getBytes(”ASCII”);

try{
  InetAddress ia = InetAddress.getByName(”www.ibiblio.org”);
  int port = 7;
  DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
  // send the packet …
}catch(IOException ex){}
```

- Veći deo vremena, najteži deo posla oko kreiranja novog DatagramPacket-a je prevođenje podataka u byte niz. Pošto ovaj fragment želi da pošalje ASCII-string, koristi getBytes() metod klase java.lang.String.
- Klasa java.io.ByteArrayOutputStream takođe može biti jako korisna za pripremu podataka za uključivanje u datagrame.

# get*() metodi

- 6 metoda za dohvatanje različitih delova datagrama: stvarnih podataka plus nekoliko polja iz zaglavlja
- ovi metodi se uglavnom koriste za datagrame primljene iz mreže
- public InetAddress getAddress() – vraća InetAddress objekat koji sadrži adresu udaljenog hosta. Ako je datagram primljen, vraća se adresa mašine koja ga je poslala. Ako je datagram kreiran lokalno da bi bio poslat udaljenoj mašini, metod vraća adresu hosta na koji je adresiran. Metod se obično koristi da odredi adresu hosta koji je poslao UDP datagram, tako da primalac može da odgovori

- public int getPort() – vraća integer koji određuje udaljeni port
- public SocketAddress getSocketAddress() – vraća SocketAddress objekat koji sadrži adresu i port udaljenog hosta. Efekat nije značajno različit od poziva getAddress() i getPort() ali uštedi jedan poziv metoda. Takođe, ako se koristi neblokirajući I/O, klasa DatagramChannel prihvata SocketAddress, ali ne i InetAddress i port.

- public byte[] getData() – vraća byte niz koji sadrži podatke datagrama. Često je potrebno konvertovati bajtove u neki drugi oblik pre nego što postanu korisni za naš program. Jedan način da se to uradi je promeniti byte niz u String korišćenjem sledećeg String konstruktora:

  public String(byte[] buffer, String encoding)

  Prvi argument, buffer, je niz bajtova koji sadrži podatke datagrama. Drugi argument sadrži ime kodiranja za string, npr. ASCII ili ISO-8859-1

  Npr. dati DatagramPacket dp primljen sa mreže, možemo konvertovati u String sa:

  String s = new String(dp.getData(), ”ASCII”);

- Ako datagram ne sadrži tekst, konvertovanje u Java podatak je komplikovanije. Jedan pristup je konvertovati byte niz koji vrati getData() u ByteArrayInputStream koristeći konstruktor:

  public ByteArrayInputStream(byte[] buffer, int offset, int length)

  buffer je byte niz koji će se koristiti kao InputStream. Važno je zadati deo buffer-a koji želimo da koristimo kao InputStream korišćenjem argumenata offset i length. Prilikom konvertovanja datagram podataka u InputStream objekte, offset je ili 0 ili dat metodom getOffset() DatagramPacket objekta a dužina je data metodom getLength() DatagramPacket objekta

npr.

```
InputStream in = new ByteArrayInputStream(packet.getData(), packet.getOffset(),
                                          packet.getLength());
```

prilikom konstruisanja ByteArrayInputStream-a moraju se zadati offset i length. NE KORISTITI ByteArrayInputStream konstruktor koji prima samo niz kao argument. Niz koji vrati packet.getData() verovatno ima u sebi i prostor koji nije popunjen podacima iz mreže. Ovaj prostor može sadržati proizvoljne slučajne vrednosti koje su se zadesile u nizu prilikom konstruisanja DatagramPacket-a

- ByteArrayInputStream može biti olančan na DataInputStream:

```
DataInputStream din = new DataInputStream(in);
```

- podaci se onda mogu čitati korišćenjem DataInputStream-ovih readInt(), readLong(), readChar() i drugih metoda.
- Naravno, ovo podrazumeva da pošiljalac koristi iste formate podataka kao Java što verovatno jeste slučaj ako je on napisan u Javi, a često je (mada ne nužno) slučaj i inače. (većina modernih računara koristi isti format pokretnog zareza kao i Java, i većina mrežnih protokola određuje cele brojeve u potpunom komplementu u mrežnom poretku bajtova, što se takođe poklapa sa Javinim formatima)

- public int getLength() – vraća broj bajtova podataka u datagramu. To nije nužno isto kao i dužina niza vraćenog sa getData(), tj. getData().length. Broj vraćen sa getLength() može biti manji od toga
- public int getOffset() – vraća poziciju u nizu vraćenom sa getData() gde počinju podaci datagrama.
- Primer 1 (DatagramExample): koristi sve ove metode da odštampa informacije iz DatagramPacket-a. Primer je malo veštački, jer program kreira DatagramPacket za koji već zna šta je unutra. Češće se ovi metodi koriste na DatagramPacket-u primljenom sa mreže, ali za to je potrebna i klasa DatagramSocket

# set*() metodi

- nekoliko metoda za promenu podataka, udaljene adrese, udaljenog porta nakon kreiranja datagrama
- ovi metodi mogu biti od značaja u situaciji kada je kreiranje i garbage collect DatagramPacket-a značajan udarac za performanse. U nekim situacijama, reusing objekata može biti značajno brži od konstruisanja novih: npr. za mrežnu igru Quake koja šalje po datagram za svaki ispaljeni metak, ili svaki centimetar pomeranja. Međutim, neophodno je koristiti veoma brzu konekciju da bi poboljšanje bilo uočljivo (jer je i sama mreža spora)

- public void setData(byte[] data) – menja koristan sadržaj UDP datagrama. Može se koristiti ako se šalje veliki fajl (veliki – veći nego što može da stane u jedan datagram) udaljenom hostu. Može se uzastopno slati isti DatagramPacket objekat, menjajući samo podatke svaki put
- public void setData(byte[] data, int offset, int length) – alternativni pristup za slanje velike količine podataka. Umesto slanja mnoštva novih nizova, svi podaci se mogu staviti u jedan niz i slati deo po deo. Primer (kod na sledećem slajdu):

```
int offset = 0;
DatagramPacket dp = new DatagramPacket(bigarray, offset, 512);
int bytesSent = 0;
while(bytesSent < bigarray.length){
  socket.send(dp);
  bytesSent += dp.getLength();
  int bytesToSend = bigarray.length – bytesSent;
  int size = (bytesToSend > 512) ? 512 : bytesToSend;
  dp.setData(bigarray, bytesSent, size);
}
```

S druge strane, ovakav pristup zahteva puno poverenja da će podaci stići, ili alternativno, zanemarivanje posledica njihovog nestizanja. Relativno je teško pridružiti redne brojeve ili druge dodatke za pouzdanost pojedinačnim paketima kada se koristi ovaj pristup

- public void setAddress(InetAddress remote) – menja adresu na koju se šalje datagram paket. Dopušta da se isti datagram pošalje većem broju različitih primalaca. Npr.

```
String s = ”Really Important Message”;
byte[] data = s.getBytes(”ASCII”);
DatagramPacket dp = new DatagramPacket(data, data.length);
dp.setPort(2000);
String network = ”128.238.5.”;
for(int host = 1; host < 255; host++){
  try{
    InetAddress remote = InetAddress.getByName(network + host);
    dp.setAddress(remote);
   socket.send(dp);
  }catch(IOException ex){
    // skip it; continue with the next host
  }
```

- Da li je ovo razuman izbor zavisi od aplikacije. Ako želite da pošaljete svim računarima u nekom delu mreže, kao što je slučaj u prethodnom fragmentu, verovatno je bolje koristiti lokalnu broadcast adresu i pustiti da mreža odradi posao. Lokalna broadcast adresa se određuje postavljanjem svih bitova IP adrese nakon ID-ova mreže i podmreže na 1. Npr. mrežna adresa Politehničkog univerziteta je 128.238.0.0. Dakle, broadcast adresa je 128.238.255.255. Slanje datagrama na 128.238.255.255 kopira ga svakom hostu na toj mreži
- Za šire rasprostranjene hostove, verovatno je bolje koristiti multicasting. Multicasting zapravo koristi istu DatagramPacket klasu koja je ovde opisana. Međutim, koristi razne IP adrese i MulticastSocket umesto DatagramSocket-a. To je opisano u poglavlju14.

- public void setPort(int port) – menja port na koji je datagram adresiran. Harold ne vidi čemu služi ovaj metod ☺ Može da se koristi za port scanner aplikaciju koja pokušava da pronađe otvorene portove koji izvršavaju određene UDP-zasnovane servise poput FSP. Druga mogućnost bila bi neka vrsta mrežne igre ili konferencijskog servera gde se klijenti koji treba da prime istu informaciju izvršavaju na različitim portovima različitih hostova. U ovom slučaju, setPort() bi se koristio sa setAddress() za promenu destinacije pre ponovnog slanja istog datagrama

- public void setAddress(SocketAddress remote) – menja adresu i port na koje se šalje datagram. Može se koristiti prilikom odgovaranja. Sledeći fragment koda prima datagram paket i odgovara na istu adresu paketom koji sadrži ASCII string ”Hello there”:

```
DatagramPacket input = new DatagramPacket(new byte[8192], 8192);
socket.receive(input);
SocketAddress address = input.getSocketAddress();
DatagramPacket output = new DatagramPacket(”Hello there”.getBytes(”ASCII”), 11);
output.setAddress(address);
socket.send(output);
```

- public void setLength(int length) – menja broj bajtova podataka u internom baferu koji se smatraju delom datagram podataka nasuprot prosto neispunjenom prostoru. Metod je koristan prilikom primanja datagrama što će biti viđeno u nastavku priče. Kada se datagram primi, njegova dužina se postavi na dužinu dolazećih podataka. Ovo znači da ako pokušamo da primimo drugi datagram u isti DatagramPacket, on je ograničen na ne više od broja bajtova u prvom. Tj. jednom kada primimo 10-bajtni datagram, svi naredni datagrami biće skraćeni na 10 bajtova, jednom kada primimo 9-bajtni, svi naredni biće skraćeni na 9-bajtova, itd. Ovaj metod omogućava da se resetuje dužina bafera tako da se naredni datagrami ne odsecaju.

# Klasa DatagramSocket

- Da bi se DatagramPacket primio ili poslao, neophodno je otvoriti datagram soket.
- U Javi, datagram soket se kreira i pristupa mu se korišćenjem DatagramSocket klase
- Svi datagram soketi su vezani za lokalni port, na kome osluškuju dolazeće podatke i koji (broj tog porta) smeštaju u zaglavlja odlazećih datagrama
- Ako pišemo klijent, nije nam bitno koji je lokalni port, pa zovemo konstruktor koji dopušta da sistem pridruži neiskorišćeni port (anonimni port). Broj porta se smešta u sve odlazeće datagrame i server će ga koristiti da bi adresirao datagrame odgovora
- Ako pišemo server, klijenti treba da znaju na kom portu server osluškuje dolazeće datagrame, pa kada server konstruiše DatagramSocket, on zadaje lokalni port na kome osluškuje.
- U svim ostalim aspektima soketi koje koriste klijenti i serveri su identični: razlikuju se samo po tome da li koriste anonimni (sistemski pridruženi) ili poznati port. Nema razlike između klijentskih i serverskih soketa, kao što je slučaj kod TCP. Ne postoji stvar kao što je DatagramServerSocket.

# Konstruktori

- DatagramSocket konstruktori se koriste u raznim situacijama, većinom kao i DatagramPacket konstruktori
- prvi konstruktor otvara datagram soket na anonimnom lokalnom portu
- drugi otvara datagram soket na poznatom portu koji osluškuje na svim lokalnim mrežnim interfejsima
- treći otvara datagram socket na poznatom portu na zadatom mrežnom interfejsu
- + konstruktor koji omogućuje zadavanje mrežnog interfejsa i porta preko SocketAddress
- Svi konstruktori rukuju samo lokalnom adresom i portom
- Udaljena adresa i port smešteni su u DatagramPacket, NE u DatagramSocket. Zapravo, jedan DatagramSocket može slati i primati datagrame od većeg broja udaljenih hostova i portova

- public DatagramSocket() throws SocketException – kreira soket koji je vezan za anonimni port
- npr.

```
try{
   DatagramSocket client = new DatagramSocket();
   // sent packets …
}catch(SocketException ex){
   System.err.println(ex);
}
```

- ovaj konstruktor treba koristiti u klijentu koji započinje konverzaciju sa serverom. U ovom scenariju, ne interesuje nas broj porta za koji je soket vezan, jer server će poslati svoj odgovor portu sa kog datagram potiče.
- Prepuštanje sistemu da pridruži port znači da ne moramo da brinemo o pronalaženju slobodnog porta. Ako, iz nekog razloga, treba da znamo broj porta, možemo ga naći metodom getLocalPort() koji je opisan kasnije u tekstu
- Isti soket može primati i datagrame koje mu server šalje nazad
- SocketException se izbacuje ako se soket ne može kreirati. Neuobičajeno je da ovaj konstruktor izbaci izuzetak; teško je zamisliti situaciju u kojoj soket ne može biti otvoren pošto sistem bira lokalni port

- public DatagramSocket(int port) throws SocketException
- kreira soket koji osluškuje dolazeće datagrame na zadatom portu
- koristite ovaj konstruktor za pisanje servera koji osluškuje na poznatom portu
- ako server osluškuje na anonimnim portovima, klijenti neće biti u mogućnosti da ga kontaktiraju
- SocketException se izbacuje ako se soket ne može kreirati. Postoje 2 uobičajena razloga za to: zadati port je već u upotrebi ili pokušavate da se konektujete na port niži od 1024, a nemate odgovarajuće privilegije (root na Unix-u)

- TCP i UDP portovi nisu u vezi. 2 servera i klijenta koji nisu u vezi mogu koristiti isti broj porta ako jedan koristi UDP, a drugi TCP.
- Primer 2: UDPPortScanner traži UDP portove koji se koriste na local hostu. Port je u upotrebi ako DatagramSocket konstruktor izbaci izuzetak
- Pretražuju se portovi počev od 1024 zbog Unix-ovog ograničenja. Lako se može proširiti i za ostale portove (ako imate root privilegije ili radite pod Windows-om)
- brzina kojom se primer izvršava jako zavisi od brzine mašine i njene UDP implementacije
- 2 minuta na prosečnom SPARCstation, manje od 12 sekundi na 1GHz TiBook, 7 sekundi na 1.4GHz Athlon sistemu pod Linux-om, i oko sat na PowerBook 5300 pod MacOS 8.
- Mnogo teže je skenirati UDP portove na udaljenoj mašini nego skenirati TCP portove. Za TCP port uvek postoji neka indikacija da je primio Vaš TCP paket. UDP ne obezbeđuje takve garancije. Da biste utvrdili da UDP server osluškuje, morate mu poslati paket koji će on prepoznati i odgovoriti na njega

- public DatagramSocket(int port, InetAddress interface) throws SocketException
- konstruktor se primarno koristi na multihomed hostovima
- kreira soket koji osluškuje dolazeće datagrame na zadatom portu i mrežnom interfejsu
- postoje 3 uobičajena razloga zbog kojih konstruktor ne uspeva: zadati port je već u upotrebi, pokušavate da se konektujete na port ispod 1024, a niste root na Unix sistemu ili interface nije adresa nijednog mrežnog interfejsa na Vašem sistemu

- public DatagramSocket(SocketAddress interface) throws SocketException
- slično kao prethodni, osim što se adresa interfejsa i port čitaju iz SocketAddress
- npr. sledeći fragment kreira soket koji sluša samo na lokalnoj loopback adresi:

```
SocketAddress address = new InetSocketAddress(”127.0.0.1”, 9999);
DatagramSocket socket = new DatagramSocket(address);
```

- protected DatagramSocket(DatagramSocketImpl impl) throws SocketException – omogućuje da potklase obezbede sopstvenu implementaciju UDP protokola ... ...

# Slanje i primanje datagrama

- Glavni zadatak klase DatagramSocket je da šalje i prima UDP datagrame. Jedan soket može i da šalje i da prima. Zapravo, on može slati i primati od većeg broja hostova istovremeno
- public void send(DatagramPacket dp) throws IOException
- nakon što se kreira DatagramPacket i DatagramSocket, paket se šalje tako što se prosledi send() metodu soketa.
- Npr. ako je theSocket objekat tipa DatagramSocket i theOutput objekat tipa DatagramPacket, theOutput se šalje koristeći theSocket sa:

```
theSocket.send(theOutput);
```

- Ako postoji problem sa slanjem podataka, izbacuje se IOException. Međutim to je ređe sa DatagramSocket-om nego sa Socket-om ili ServerSocket-om, pošto nepouzdana priroda UDP-a znači da ne želite izbacivanje izuzetka samo zato što paket nije prispeo na odredište.
- IOException možete dobiti ako pokušate da pošaljete veći datagram nego što native mrežni software hosta podržava, ali možda i ne dobijete izuzetak zbog toga. To jako zavisi od native UDP software-a OS-a i native koda između ovog i Javine klase DatagramSocketImpl.
- Ovaj metod takođe može izbaciti SecurityException ako nam SecurityManager ne dopusti da komuniciramo sa hostom na koji je paket adresiran. Ovo je primarno problem za aplete i drugi “remotely” loaded kod.

- Primer 3: UDP-zasnovan discard klijent.
- Čita linije koje korisnik unosi iz System.in i šalje ih discard serveru, koji prosto odbacuje sve podatke. Svaka linija se smešta u DatagramPacket. Mnogi jednostavniji Internet protokoli, poput discard, imaju i TCP i UDP implementaciju

# primer 3 objašnjenja

- klasa ima jedno statičko polje, DEFAULT_PORT, koje je postavljeno na standardni port za discard protokol (port 9) i jedan metod, main()
- main() metod čita hostname iz komandne linije i konvertuje taj hostname u InetAddress objekat server.
- BufferedReader je olančan na System.in kako bi čitao ulaz korisnika sa tastature
- Zatim se konstruiše objekat theSocket tipa DatagramSocket. Po kreiranju soketa, program ulazi u beskonačnu while-petlju koja čita unos korisnika liniju po liniju koristeći readLine(). Pošto discard protokol radi samo sa neobrađenim bajtovima, možemo ignorisati probleme sa kodiranjem karaktera

- U while petlji, svaka linija se konvertuje u byte niz korišćenjem metoda getBytes() i bajtovi se smeštaju u DatagramPacket theOutput. Konačno, theOutput se šalje preko theSocket, i petlja se nastavlja.
- Ako u bilo kom trenutku korisnik u liniji ukuca samo tačku, program se završava.
- DatagramSocket konstruktor može izbaciti SocketException, koji mora biti uhvaćen. Pošto je ovo discard klijent, ne moramo brinuti o podacima koji se vraćaju od servera

# public void receive(DatagramPacket dp) throws IOException

- Metod prima jedan UDP datagram sa mreže i smešta ga u DatagramPacket objekat dp koji treba da postoji pre poziva metoda
- Poput metoda accept() klase ServerSocket, ovaj metod blokira pozivajuću nit dok datagram ne stigne. Ako Vaš program radi još nešto osim što čeka datagrame, receive() treba pozivati u posebnoj niti

- Bafer datagrama treba da bude dovoljno velik za smeštanje primljenih podataka
- Ako to nije slučaj, receive() smešta u bafer onoliko podataka koliko u njega može da stane, ostalo se gubi.
- može biti od koristi da se upamti da je maksimalna veličina dela podataka UDP datagrama 65507 bajtova (65536 = maksimalna veličina IP datagrama minus 20 bajtova = veličina IP zaglavlja, i minus 8 bajtova = veličina UDP zaglavlja). Neki application protokoli koji koriste UDP postavljaju dalja ograničenja na maksimalan broj bajtova u paketu; npr. NFS koristi pakete maksimalne veličine 8192 bajtova

- Ako postoji problem u primanju podataka, može biti izbačen IOException. U praksi, to je retko.
- Za razliku od send(), ovaj metod ne izbacuje SecurityException ako aplet primi datagram od nekog drugog hosta. Međutim, tiho odbacuje sve takve podatke.

- Primer 4: UDP discard server koji prima dolazeće datagrame. Samo iz zabave, loguje podatke svakog datagrama u System.out tako da možemo videti ko šta šalje Vašem discard serveru
- klasa je jednostavna. Ima jedan metod, main(). On čita iz komandne linije port na kome server osluškuje. Ako port nije zadat u komandnoj liniji, server osluškuje na portu 9. Zatim otvara DatagramSocket na tom portu i kreira DatagramPacket sa baferom veličine 65507 bajtova – dovoljno velik da primi svaki mogući paket.
- Server zatim ulazi u beskonačnu petlju koja prima pakete i štampa sadržaj i host sa koga potiču u konzoli.
- Discard server visokih performansi bi preskočio ovaj korak
- Kada se datagram primi, dužina paketa se postavlja na dužinu podataka u tom datagramu. Iz tog razloga, u poslednjem koraku petlje, dužina paketa se resetuje na maksimalnu moguću vrednost. Inače, dolazeći paketi bi bili ograničeni na minimalnu veličinu svih prethodnih datagrama.
- Možemo pokrenuti discard klijent na jednoj mašini i konektovati na discard server na drugoj mašini kako bismo proverili da mreža radi.

# public void close()

- Poziv metoda close() za objekat klase DatagramSocket oslobađa port koji zauzima taj soket.
- Npr.

```
try{
   DatagramSocket server = new DatagramSocket();
   server.close();
}catch(SocketException ex){
   System.err.println(ex);
}
```

- Nikada nije loša ideja zatvoriti DatagramSocket kada smo završili sa njim. Posebno je važno zatvoriti nepotreban soket ako će program nastaviti da se izvršava još prilično dugo. Npr. u primeru 2 (UDPPortScanner) close() metod je od ključnog značaja. Da ovaj program ne zatvara sokete koje otvara, zauzeo bi svaki UDP port sistema prilično dugo. S druge strane, ako se program završava uskoro nakon što završimo rad sa DatagramSocket-om, ne moramo eksplicitno da zatvorimo soket, soket se automatski zatvara.
- Zatvaranje nepotrebnih soketa je dobra programerska praksa

# public int getLocalPort()

- metod vraća int koji predstavlja lokalni port na kome soket osluškuje. Ovaj metod se koristi ako smo kreirali DatagramSocket sa anonimnim portom i želimo da saznamo koji port je pridružen soketu
- npr:

```
try{
   DatagramSocket ds = new DatagramSocket();
   System.out.println(”The socket is using port ” +
   ds.getLocalPort();
}catch(SocketException ex){
   ex.printStackTrace();
}
```

- public InetAddress getLocalAddress() – vraća InetAddress objekat koji predstavlja lokalnu adresu za koju je soket vezan. Retko je potreban u praksi. Obično ili znate ili Vas ne zanima.
- public SocketAddress getLocalSocketAddress() – vraća SocketAddress objekat koji u sebi enkapsulira lokalni interfejs i port za koje je soket vezan

# Managing Connections

- Za razliku od TCP soketa, datagram soketi nisu vrlo izbirljivi po pitanju toga sa kim komuniciraju. Zapravo, podrazumevano oni komuniciraju sa svakim, ali to često nije ono što mi želimo. Npr. apletima je jedino dopušteno da šalju datagrame i primaju datagrame od svog hosta.
- NFS ili FSP klijent treba da prihvata pakete samo od servera sa kojim komuniciraju
- Mrežna igra treba da osluškuje datagrame samo od osoba koje igraju igru
- Postoji 4 metoda koji dopuštaju da izaberemo hostove kojima možemo da šaljemo i od kojih želimo da primamo datagrame, a da odbijamo sve ostale pakete

- public void connect(InetAddress host, int port) – ne uspostavlja zaista konekciju u TCP smislu. Međutim, određuje da će DatagramSocket slati i primati pakete samo od zadatog udaljenog hosta na zadatom udaljenom portu. Pokušaji slanja paketa nekom drugom hostu ili portu izbaciće IllegalArgumentException. Paketi primljeni od drugog hosta ili drugog porta  biće odbačeni bez izuzetka ili drugog obaveštenja
- kada se pozove connect() vrši se sigurnosna provera. Ako je VM dopušteno da šalje podatke tom hostu i portu, provera prolazi tiho. Inače, izbacuje se SecurityException. Međutim, nakon što se napravi konekcija, send() i receive() na DatagramSocket-u ne vrše više sigurnosne provere koje uobičajeno prave

- public void disconnect() – raskida ”konekciju” tako da DatagramSocket može ponovo da šalje i prima pakete od bilo kog hosta i bilo kog porta
- public int getPort() – ako i samo ako je DatagramSocket connected, ovaj metod vraća udaljeni port na koji je konektovan. Inače, vraća -1

- public InetAddress getInetAddress() – ako i samo ako je soket konektovan, metod vraća adresu udaljenog hosta na koji je konektovan. Inače, vraća null
- public InetAddress getRemoteSocketAddress() – isto kao i prethodno
- SOCKET OPTIONS ... (page 22 of 58)

# SO_TIMEOUT

- SO_TIMEOUT je količina vremena, u milisekundama, koju receive() čeka na dolazeći datagram pre nego što izbaci InterruptedIOException (potklasa od IOException). Vrednost mora biti nenegativna. Ako je 0, receive() ne istekne nikada. Ova vrednost može se promeniti metodom setSoTimeout() i dohvatiti metodom getSoTimeout():
- public synchronized void setSoTimeout(int timeout) throws SocketException
- public synchronized int getSoTimeout() throws IOException
- podrazumevano, vrednost je 0, a ima nekoliko situacija u kojima želimo da postavimo SO_TIMEOUT. Npr. ako implementiramo secure protokol koji zahteva da se odgovori dese u nekom zadatom vremenskom intervalu. Takođe, možemo odlučiti da je host sa kojim komuniciramo mrtav (nedostupan ili ne odgovara) ako ne primimo odgovor pre nego što istekne određena količina vremena
- Metod setSoTimeout() postavlja SO_TIMEOUT polje datagram soketa. Kada timeout istekne, izbacuje se InterruptedIOException.
- Ova opcija se postavlja pre poziva receive(). Ne može se promeniti dok receive() čeka na datagram. timeout argument mora biti veći ili jednak od 0, ako nije, setSoTimeout() izbacuje SocketException
- Npr.

```
try{
  buffer = new byte[2056];
  DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
  DatagramSocket ds = new DatagramSocket(2048);
  ds.setSoTimeout(30000); // block for no more than 30 seconds
  try{
    ds.receive(dp);
    // process the packet …
  }catch(InterruptedIOException ex){
    ds.close();
    System.err.println(”No connection within 30 seconds”);
  }catch(SocketException ex){
     System.err.println(ex);
  }catch(IOException ex){
     System.err.println(“Unexpected IOException: “ + ex);
  }
}
```

- getSoTimeout() metod vraća tekuću vrednost SO_TIMEOUT polja DatagramSocket objekta

```
public void printSoTimeout(DatagramSocket ds){
  int timeout = ds.getSoTimeout();
  if(timeout > 0)
    System.out.println(ds + ” will time out after ” + timeout +
  ”milliseconds.”);
  else if (timeout == 0)
    System.out.println(ds + ” will never time out.”);
  else
    System.out.println(”Something is seriously wrong with ” + ds);
}
```

# Neke korisne aplikacije

- Nekoliko Internet servera i klijenata koji koriste DatagramPacket i DatagramSocket
- Neki od njih su slični sa ranijim primerima, jer mnogi Internet protokoli imaju i TCP i UDP implementacije
- Kada host primi IP paket, host određuje da li je paket TCP ili UDP istraživanjem IP zaglavlja.
- Kao što je ranije rečeno, ne postoji veza između UDP i TCP portova; TCP i UDP serveri mogu deliti iste brojeve portova bez problema. Po dogovoru, ako servis ima i TCP i UDP implementaciju, koristi se isti port za obe, mada nema tehničkih razloga da to bude tako

# Jednostavni UDP klijenti

- Nekoliko Internet servisa treba da zna samo adresu i port klijenta i ignoriše sve podatke koje klijenti pošalju u datagramima
- Daytime, time i chargen su takvi protokoli
- Svaki od njih odgovara na neki način, bez obzira na podatke koji su u datagramu, zapravo, bez obzira ima li bilo kakvih podataka u datagramu
- Klijenti za ove protokole prosto šalju UDP datagram serveru i čitaju odgovor koji se vrati.
- Počinjemo sa jednostavnim klijentom UDPPoke koji šalje prazan UDP paket zadatom hostu i portu i čita odgovor od tog hosta.

# primer 5 - UDPPoke

- Primer 5 (UDPPoke): klasa UDPPoke ima 3 privatna polja: bufferSize – zadaje kolika je veličina paketa koji se očekuje kao odgovor. Bafer veličine 8192 bajta je dovoljno velik za većinu protokola za koje je UDPPoke koristan, ali može se i povećati prosleđivanjem veće vrednosti konstruktoru. DatagramSocket objekat socket će se koristiti i za slanje i za primanje datagrama. Konačno, DatagramPacket objekat outgoing je poruka koja se šalje pojedinačnim serverima

- Konstruktori inicijalizuju sva 3 polja koristeći InetAddress za host i int-ove za port, dužinu bafera i broj milisekundi koliko se čeka. Ova poslednja 3 postaju deo DatagramSocket polja socket. Ako dužina bafera nije zadata, koristi se 8192. Ako timeout nije zadat, 30 sekundi (30000 milisekundi) se koristi. Host, port i veličina bafera se takođe koriste za konstruisanje outgoing DatagramPacket-a.
- Iako bi teoretski trebalo da bude moguće poslati datagram bez podataka, bug-ovi u nekim Java implementacijama zahtevaju da dodamo bar jedan bajt podataka datagramu.
- Jednostavni serveri koje trenutno razmatramo ignorišu ove podatke

# poke() metod

- Nakon što je UDPPoke objekat konstruisan, klijenti pozivaju njegov poke() metod da pošalju prazan outgoing datagram odredištu i pročitaju odgovor
- odgovor se inicijalno postavlja na null
- kada se očekivani datagram pojavi, njegovi podaci se kopiraju u polje response
- metod vraća null ako odgovor ne dođe dovoljno brzo ili ne dođe uopšte

# main()

- main() metod prosto čita host i port na koje se treba konektovati iz komandne linije, konstruiše UDPPoke objekat i poziva poke() metod.
- Većina jednostavnih protokola kojima odgovara ovaj klijent vraća ASCII tekst, tako da pokušavamo da konvertujemo odgovor u ASCII string i štampamo ga.
- Ne podržavaju sve VM ASCII kodiranje karaktera, pa obezbeđujemo mogućnost korišćenja ASCII nadskupa Latin-1 (8859-1) kao backup

- kada imamo klasu UDPPoke, UDP daytime, time, chargen i quote of the day klijenti su skoro trivijalni.
- Primer 6: time client
- najkomplikovaniji deo je konvertovanje 4 neobrađena bajta koja je vratio server u java.util.Date objekat. Korišćen je isti algoritam kao za TCP verziju klijenta
- ostali protokoli se ostavljaju za vežbu

# UDP Server

- Klijenti nisu jedini programi koji dobijaju benefit od reusable implementacije. Serveri za ove protokole su veoma slični.
- Svi oni čekaju UDP datagrame na zadatom portu i odgovarju na svaki datagram drugim datagramom.
- Serveri se razlikuju samo po sadržaju datagrama koje vraćaju

# primer 7: UDPServer

- jednostavna klasa koja se može naslediti da obezbedi specifične servere za različite protokole
- UDPServer klasa ima dva polja: int bufferSize i DatagramSocket socket, od kojih je drugo protected pa ga potklase mogu koristiti
- Konstruktor otvara DatagramSocket socket na zadatom lokalnom portu kako bi primao datagrame ne veće od bufferSize bajtova
- UDPServer implementira Runnable tako da se veći broj instanci može paralelno izvršavati. Njegov run() metod sadrži beskonačnu petlju koja uzastopno prima dolazeći datagram i odgovara prosleđujući ga apstraktnom respond() metodu. Ovaj metod će biti predefinisan odgovarajućim potklasama u cilju implementiranja različitih vrsta servera
- UDPServer je veoma fleksibilna klasa. Potklase mogu poslati 0, 1 ili mnogo datagrama kako bi odgovorile na svaki datagram koji dođe.
- Ako je neophodno dosta procesiranja kako bi se odgovorilo na paket, respond() metod se može poslati u nit kako bi odradio to. Međutim, UDP serveri teže da nemaju produžene interakcije sa klijentom. Svaki paket koji dođe tretira se nezavisno od ostalih paketa, tako da se odgovorom obično rukuje direktno u metodu respond(), bez generisanja niti.

# primer 8: FastUDPDiscardServer

- Najlakši protokol za obradu je discard
- sve što treba napisati je main() metod koji postavlja port i startuje nit.
- respond() je metod koji ne radi ništa
- primer je UDP discard server visokih performansi koji ne radi ništa sa dolazećim paketima

# primer 9: LoggingUDPDiscardServer

- za nijansu interesantniji discard server koji štampa dolazeće pakete u System.out

## primer 10: UDPEchoServer

- Nije mnogo teže implementirati echo server. Za razliku od stream-based TCP echo servera, ovde nije potreban veći broj niti da bi se rukovalo većim brojem klijenata

## primer 11: UDPDaytimeServer

- samo neznatno komplikovaniji. Server osluškuje UDP datagrame na portu 13. Kada detektuje dolazeći datagram, vraća tekući datum i vreme na serveru kao jednoredni ASCII string.

# primer 12: UDP Echo Client

- klasa UDPPoke implementirana ranije nije pogodna za sve protokole. Posebno, protokoli koji zahtevaju veći broj datagrama zahtevaju drugačiju implementaciju
- Echo protokol ima i TCP i UDP implementaciju
- Implementiranje echo protokola TCP-jem je jednostavno. Sa UDP-om je komplikovanije jer nemamo I/O tokove niti koncept konekcije
- TCP zasnovan echo klijent može poslati poruku i čekati na odgovor na istoj konekciji. Međutim, UDP zasnovan echo klijent nema garancije da je poruka koju je poslao primljena. Zato ne može prosto čekati na odgovor, mora biti spreman da šalje i prima podatke asinhrono

- Korišćenjem niti ovo ponašanje je prilično jednostavno za implementaciju.
- Jedna nit može da procesira ulaz korisnika i šalje ga echo serveru, dok druga nit prihvata ulaz od servera i prikazuje ga korisniku.
- Klijent je podeljen u 3 klase: glavnu UDPEchoClient, SenderThread klasu i ReceiverThread klasu

- UDPEchoClient klasa treba da izgleda poznato. Čita hostname iz komandne linije i konvertuje ga u InetAddress objekat.
- UDPEchoClient koristi ovaj objekat i podrazumevani echo port da konstruiše SenderThread objekat. Konstruktor može izbaciti SocketException, pa se on hvata. Zatim startuje SenderThread. Isti DatagramSocket koji koristi SenderThread koristi se za konstruisanje ReceiverThread, koja se zatim startuje.
- Važno je koristiti isti DatagramSocket i za primanje i za slanje podataka jer će echo server poslati odgovor nazad na port sa koga su podaci poslati

# klasa SenderThread

- SenderThread klasa čita ulaz iz konzole, liniju po liniju, i šalje to echo serveru.
- ulaz se obezbeđuje sa System.in, ali drugačiji klijent bi mogao da uključi opciju čitanja iz drugog toka – možda otvaranjem FileInputStream da čita iz fajla.
- 3 polja ove klase definišu server kome se šalju podaci, port na tom serveru i DatagramSocket koji vrši slanje, sve se postavlja u jednom konstruktoru
- DatagramSocket je konektovan na udaljeni server kako bi se osigurao da su svi primljeni datagrami zapravo poslati od strane pravog servera. Prilično je neverovatno da će neki drugi server sa Interneta bombardovati ovaj port čudnim podacima. Međutim, dobra je navika osigurati da paketi koje primamo dolaze sa pravog mesta, posebno ako je sigurnost bitna

- metod run() procesira ulaz liniju po liniju.
- Da bi ovo uradio, BufferedReader userInput je olančan na System.in.
- beskonačna petlja čita linije korisnikovog ulaza.
- svaka linija se smešta u theLine.
- sama tačka u liniji označava kraj ulaza korisnika i prekida petlju
- inače, bajtovi podataka se smeštaju u niz korišćenjem getBytes() metoda klase java.lang.String.
- Dalje, niz se smešta u koristan deo DatagramPacket-a output, zajedno sa informacijom o serveru, portu i dužinom podataka.
- Ovaj paket se šalje svom odredištu korišćenju socket-a.
- Ova nit zatim radi yield da da šansu drugim nitima da se izvršavaju

# klasa ReceiverThread

- ReceiverThread klasa čeka da stignu datagrami iz mreže.
- Kada se datagram primi, konvertuje se u String i štampa na System.out kako bi se prikazao korisniku. Napredniji EchoClient može uključiti opciju slanja izlaza negde drugde
- Klasa ima 2 polja. Važnije je DatagramSocket theSocket, koje mora biti isti DatagramSocket koji koristi SenderThread. Podaci stižu na port koji koristi taj DatagramSocket. Bilo kom drugom DatagramSocket-u ne bi bilo dopušteno da se konektuje na isti port. Drugo polje, stopped, je boolean koji se koristi da prekine ovu nit bez poziva stop() metoda

- run() metod je beskonačna petlja koja koristi receive() metod socket-a da čeka na dolazeće datagrame
- kada se pojavi dolazeći datagram, konvertuje se u String iste dužine kao što su dolazeći podaci i štampa na System.out.
- Kao i ulazna nit, ova nit zatim radi yield kako bi dala šansu i drugim nitima da se izvršavaju

# DatagramChannel

- Klasa se koristi u neblokirajućim UDP aplikacijama
- Može biti registrovana Selector-om. To je korisno u serverima gde jedna nit može obrađivati komunikaciju sa više različitih klijenata. Međutim, UDP je po svojoj prirodi mnogo više asinhron nego TCP, pa je efekat manji

- U UDP je uvek slučaj da jedan datagram soket može procesirati zahteve većeg broja klijenata i za ulazom i za izlazom
- DatagramChannel klasa dodaje mogućnost da se to čini na neblokirajući način, pa metodi vraćaju kontrolu brzo ako mreža nije momentalno spremna da primi ili pošalje podatke

# Korišćenje DatagramChannel

- Koristi se i klasa DatagramSocket kako bi se kanal vezao za port. Međutim, kasnije se više ne mora koristiti, kao ni DatagramPacket, koja se ne koristi uopšte.
- Čitaju se i pišu ByteBuffer objekti, kao sa SocketChannel.

# Otvaranje soketa

- Klasa ne poseduje javne konstruktore
- Novi DatagramChannel objekat se kreira korišćenjem statičkog open() metoda:

```
public static DatagramChannel open() throws IOException
```

Npr.

```
DatagramChannel channel = DatagramChannel.open();
```

- Kanal inicijalno nije vezan ni za jedan port
- Da bismo ga vezali, treba da pristupimo kanalovom peer objektu tipa DatagramSocket, koristeći metod socket():

```
public abstract DatagramSocket socket()
```

Npr. da vežemo channel za port 3141:

```
SocketAddress address=new InetSocketAddress(3141);
DatagramSocket socket = channel.socket();
socket.bind(address);
```

# Povezivanje (konektovanje)

- Poput DatagramSocket, i DatagramChannel može biti konektovan, tj. može biti konfigurisan tako da prima datagrame od i šalje datagrame isključivo jednom hostu. To se postiže metodom

public abstract DatagramChannel connect(SocketAddress remote) throws IOException

- Ipak, za razliku od connect() metoda klase SocketChannel, ovaj metod zapravo ne šalje niti prima pakete preko mreže, jer je UDP connectionless protokol.
- Zato ovaj metod brzo vraća kontrolu, ne blokira.
- Postoji isConnected() metod koji vraća true akko je DatagramSocket konektovan:

```
public abstract boolean isConnected()
```

(kaže da li je DatagramChannel ograničen na 1 hosta)

- Za razliku od SocketChannel, DatagramChannel ne mora da bude konektovan da bi slao ili primao podatke
- disconnect() metod raskida konekciju:

public abstract DatagramChannel disconnect() throws IOException

dopušta da kanal ponovo može da šalje i prima podatke od većeg broja hostova

- Konektovani kanali mogu biti brži od nekonektovanih kod apleta jer VM mora da proveri da li je konekcija dopuštena samo prilikom inicijalnog poziva connect() metoda, a ne svaki put kada se primi ili pošalje paket.

# Primanje (receiving)

- Metod receive() čita 1 datagram paket iz kanala u ByteBuffer. Vraća adresu hosta koji je poslao paket:

  public abstract SocketAddress receive(ByteBuffer dst) throws IOException

- Ako je kanal blokirajući (podrazumevano), metod neće vratiti kontrolu dok ne pročita paket
- Ako je kanal neblokirajući, metod neposredno vraća null ako paket nije dostupan za čitanje

- Ako datagram paket sadrži više podataka nego što može da stane u bafer, višak podataka se odbacuje bez obaveštenja o problemu. Ne dobijamo BufferOverflowException niti bilo šta slično. Na kraju krajeva, UDP je nepouzdan.
- Ovakvo ponašanje uvodi novi nivo nepouzdanosti. Podaci mogu bezbedno prispeti iz mreže i biti izgubljeni u našem sopstvenom programu

# Primer: discard server

- Koristeći metod receive() ponovo implementiramo discard server koji loguje host koji je poslao podatke, kao i same podatke
- Primer izbegava potencijalni gubitak podataka koristeći bafer koji je dovoljno velik da sadrži UDP paket i čisteći ga pre ponovne upotrebe

# Slanje (sending)

- Metod send() piše 1 datagram paket iz ByteBuffer u kanal za adresu zadatu kao drugi argument:
  public abstract int send(ByteBuffer src, SocketAddress target) throws IOException
- Izvor ByteBuffer može biti ponovo iskorišćen ako želimo da iste podatke pošaljemo većem broju klijenata. Samo ne smemo da zaboravimo da ga najpre premotamo (rewind)

- Metod send() vraća broj ispisanih bajtova
- To će biti ili broj bajtova preostalih u izlaznom baferu ili 0
- 0 je ako nema dovoljno prostora u izlaznom baferu mrežnog interfejsa za količinu podataka koju pokušamo da pošaljemo
- Ako stavimo više podataka u bafer nego što mrežni interfejs može da prihvati, on nikada ništa neće poslati.
- Ovaj metod ne deli podatke u veći broj paketa. On piše sve ili ništa.

# Primer: jednostavan echo server zasnovan na kanalima

- Metod receive() čita paket, uglavnom kao i u prethodnom primeru. Međutim, sada, radije nego da loguje paket na System.out, vraća iste podatke klijentu koji ih je poslao
- Ovaj program je blokirajući i sinhron.  To je mnogo manji problem za UDP-protokole nego za TCP. Server ne mora i ne čeka da klijent bude spreman da primi podatke

# Reading

- Osim metoda receive() specijalne namene, DatagramChannel ima i uobičajena 3 read() metoda (prvi prima jedan argument tipa ByteBuffer, drugi jedan argument tipa ByteBuffer[], a treći tri argumenta: prvi tipa ByteBuffer[], a zatim i dva int-a: offset i length
- Ovi metodi se mogu koristiti samo na konektovanim kanalima, tj. pre poziva nekog od ovih metoda mora se pozvati connect() da “zalepi” kanal za određeni udaljeni host

- Ovo ih čini pogodnijim za korišćenje od strane klijenata koji znaju sa kim će komunicirati, nego za servere koji moraju prihvatati ulaz od mnogih hostova istovremeno, a koji normalno nisu poznati pre pristizanja prvog paketa od njih
- Svaki od ova 3 read() metoda čita 1 datagram paket iz mreže. Što je moguće više podataka iz tog datagrama smešta se u argument ByteBuffer(s).
- Svaki read() metod vraća broj pročitanih bajtova ili -1 ako je kanal zatvoren. Može vratiti 0 iz jednog od sledećih razloga:
  - kanal je neblokirajući i nema spremnih paketa
  - datagram paket ne sadrži podatke
  - bafer je pun

- Kao i sa receive() metodom, ako datagram paket ima više podataka nego što ByteBuffer(s) može da sadrži, višak podataka se odbacuje bez informacije o problemu

# Writing

- DatagramChannel ima 3 write() metoda koja se mogu koristiti umesto metoda send()
- Prvi prima 1 argument tipa ByteBuffer, drugi 1 argument tipa ByteBuffer[], a treći tri argumenta: prvi tipa ByteBuffer[], a zatim dva int-a: offset i length
- Ovi metodi mogu biti korišćeni samo sa konektovanim kanalima, inače ne znaju gde da pošalju paket

- Svaki od ovih metoda šalje 1 datagram paket preko konekcije. Nijedan od njih ne garantuje da će ispisati kompletan sadržaj bafera. Međutim kursor-zasnovana priroda bafera nam omogućuje da zovemo ovaj metod iznova i iznova sve dok se bafer potpuno ne isprazni i podaci svi pošalju, možda koristeći veći broj datagram paketa:

```
while(buffer.hasRemaining() && channel.write(buffer)!=-1);
```

# Primer: UDP echo klijent

- Možemo koristiti read() i write() metode da implementiramo jednostavni UDP echo klijent
- Na klijentskoj strani, jednostavno je izvršiti konektovanje pre slanja.
- Pošto paketi mogu biti izgubljeni prilikom slanja (zapamtiti uvek da je UDP nepouzdan), ne želimo da odlažemo slanje dok čekamo da primimo paket
- Zato možemo da iskoristimo prednost selektora i neblokirajućeg I/O. Oni funkcionišu za UDP isto kao i za TCP. Ovog puta, međutim, umesto slanja tekstualnih podataka, pošaljimo 100 int-ova od 0 do 99. Štampaćemo vraćene vrednosti tako da je jednostavno otkriti ako je neki paket izgubljen.

- Postoji 1 ključna razlika između selektovanja TCP kanala i selektovanja datagram kanala. Pošto su datagram kanali čisto connectionless (uprkos connect() metodu), moramo da primetimo kada je transfer podataka gotov i da završimo.
- U ovom primeru, pretpostavljamo da su podaci gotovi kada su svi paketi poslati i 1 minut je prošao otkako je poslednji paket primljen. Svi očekivani paketi koji nisu pristigli do tog momenta smatraju se izgubljenim

# Zatvaranje

- Kao i sa regularnim datagram soketima, kanal treba zatvoriti kada smo završili sa njim kako bi se oslobodio port, i drugi, eventualno korišćeni, resursi

  public void close() throws IOException

- Zatvaranje već zatvorenog kanala nema efekta
- Pokušaj pisanja podataka ili čitanja podataka iz zatvorenog kanala izbacuje izuzetak

  public boolean isOpen()