Razred `java.net.InetAddress`:

- model adrese računala na Internetu
- primjerci razreda pamte par (*simboličko ime*, *binarni zapis adrese*)
- imamo dva podrazreda:
  - `java.net.Inet4Address`: model IPv4, adresa je polje okteta duljine 4
  - `java.net.Inet6Address`: model IPv6, adresa je polje okteta duljine 16 (4 x dulja od v4)
- statičke metode nude stvaranje primjeraka:
  - iz polja okteta: `InetAddress.getByAddress(byte[] addr): InetAddress`,
  - iz stringa koji predstavlja tekstovni zapis IP adrese (tipa: "161.53.65.224"): `InetAddress.getByName("java.zemris.fer.hr")`
  - iz stringa koji predstavlja simboličko ime računala (pretvorba putem DNS-a): `InetAddress.getByName("java.zemris.fer.hr")`
  - ...

–-- Primjer1.java –--

Razred java.net.NetworkInterface:

- model mrežnog sučelja (kartice)
- statičke metode za dohvat po indeksu, imenu te enumeriranje
- članske metode za dohvat informacija o konkretnom sučelju

–-- Primjer2.java –--

Razred `java.net.SocketAddress`:

- model adrese aplikacije na Internetu: uređeni par (*adresa računala*, *port*)
- port je 16 bitni podatak,
  - vrijednosti su od 0 do 65535
  - 0 ima posebno značenje (jocker) – specificiramo kada ne želimo zadati neki konkretan port već želimo prepustiti operacijskom sustavu da nam po potrebi dodijeli neki nekorišteni
- imamo podrazred `java.net.InetSocketAddress`:
  - uređeni par IP adrese i porta
  - konstruktor `InetSocketAddress(int port)` koristi "jocker" adresu i zadani port
    - aplikacija koja će slušati na tako konfiguriranoj adresi primat će promet sa svih adresa računala i zadanog porta
  - konstruktor `InetSocketAddress(InetAddress addr, int port)` koristi predanu adresu i port
    - ako računalo ima više IP adresa, aplikacija koja će slušati na tako konfiguriranoj adresi primat će promet poslan samo specificiranoj adresi i zadanom portu
  - konstruktor `InetSocketAddress(String hostname, int port)` koristi IP adresu koju dobije razješavanjem predanog stringa te zadani port
    - hostname mora biti jedno od imena računala na kojem se aplikacija pokreće
    - ako računalo ima više IP adresa, aplikacija koja će slušati na tako konfiguriranoj adresi primat će promet poslan samo utvrđenoj adresi i zadanom portu

Protokol UDP

- *bespojna* usluga, nepouzdan prijenos, protokol prijenosnog sloja (4. sloj ISO/OSI referentnog modela)
- korisnik sam stvara pakete (sam brine o njihovom formatu/sadržaju) i šalje ih
- nema nikakve garancije da će paket biti isporučen, ako ih se šalje više, da će stići redoslijedom slanja ili da jedan paket neće biti isporučen više puta
- paket je modeliran razredom `DatagramPacket`:
  - enkapsulira spremnik s podatcima koje treba poslati / spremnik u koji će podatci biti zapisani ako ga koristimo za primanje (višak pristiglih podataka se gubi ako je spremnik premali)
  - omogućava definiranje kamo se paket šalje / očitavanja od kuda je stigao (IP, port)
- pristupna točka koju koristimo za primanje/slanje modelirana je razredom `DatagramSocket`
  - poslužitelj je stvara tako da obavezno definira port na kojem želi prihvaćati promet (zašto je to važno?)
  - klijent je može stvoriti bez specificirane adrese i porta (u tom slučaju operacijski sustav dodjeljuje adresu koja će se koristiti pri slanju te neki od slobodnih portova)

–-- Klijent.java / Posluzitelj.java –--

protokol TCP

- *spojna* usluga, pouzdan prijenos, protokol prijenosnog sloja (4. sloj ISO/OSI referentnog modela)
- po uspostavi spoja svaka pristupna točka ima uspostavljena dva binarna toka (engl. *stream*) prema onoj drugoj pristupnoj točki: jedan koristi za pisanje podataka koji se potom prenose na drugu stranu a drugi za čitanje podataka koji su pristigli s druge strane
  - tokovi se dobivaju pozivima `getInputStream()` i `getOutputStream()`
- **pristupna točka poslužitelja** modelirana je razredom `ServerSocket`
  - prilikom stvaranja, predaje se port na kojem je potrebno slušati (može i adresa)
  - poslužitelj u petlji prihvaća veze metodom `accept()`
    - od metode `accept()` dobiva referencu na novu pristupnu točku razreda `Socket` preko koje može razgovarati sa spojenim klijentom uporabom tokova za čitanje i pisanje
    - kad je gotov, tu pristupnu točku zatvori i vraća se na čekanje na prihvat veza novim pozivom metode `accept()`
    - alternativno, obrada klijenta može se prepustiti novoj dretvi a glavni program se odmah vraća novom pozivu metode `accept()`
- **pristupna točka klijenta** modelirana je razredom `Socket`
  - prilikom stvaranja u konstruktoru se predaju i adresa i port udaljene aplikacije na koju se želimo spojiti uporabom protokola TCP
  - obavlja se 3-way handshake, i jednom kad je veza uspostavljena, klijent može koristiti tokove za čitanje i pisanje okteta

U nastavku razmatramo protokol HTTP.

–-- Klijent.java / Posluzitelj.java –--