Objektno orijentirano programiranje

2: Kreiranje objekata. Stringovi, polja. Memorija programa (*stack, heap*)

Creative Commons

You are free to

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

under the following terms

- **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **NonCommercial** — You may not use the material for commercial purposes.
- **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

- https://creativecommons.org/licenses/by-nc-sa/4.0/

IDE i Java projekti

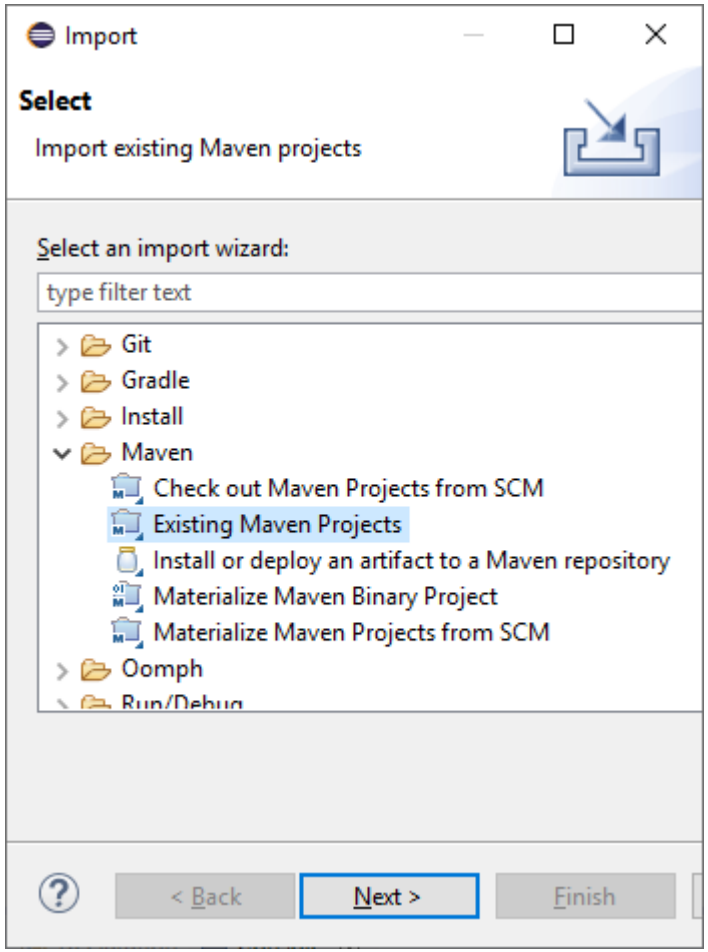
- Uvodni primjeri su koristili mape *src* and *bin* i IDE nije bio korišten
  - samo *src* je sadržan u repozitoriju
- Izvorni kod nije sadržavao specifične datoteke pojedinog IDE-a
  - različiti za Eclipse, NetBeans, itd…
  - za svaki primjer treba kreirati novi projekti, kopirati izvorni kod, postaviti ovisnosti, postavke prevodioca, … → prilično nepraktično
- Rješenje:
  - korištenje nekog od (većine IDE-a podržanih) razvojnih okvira za definiranje i automatizaciju prevođenja Java projekata.
  - *Apache Maven* je jedan od uobičajenih alata
    - Alternative: Gradle, Ant, …

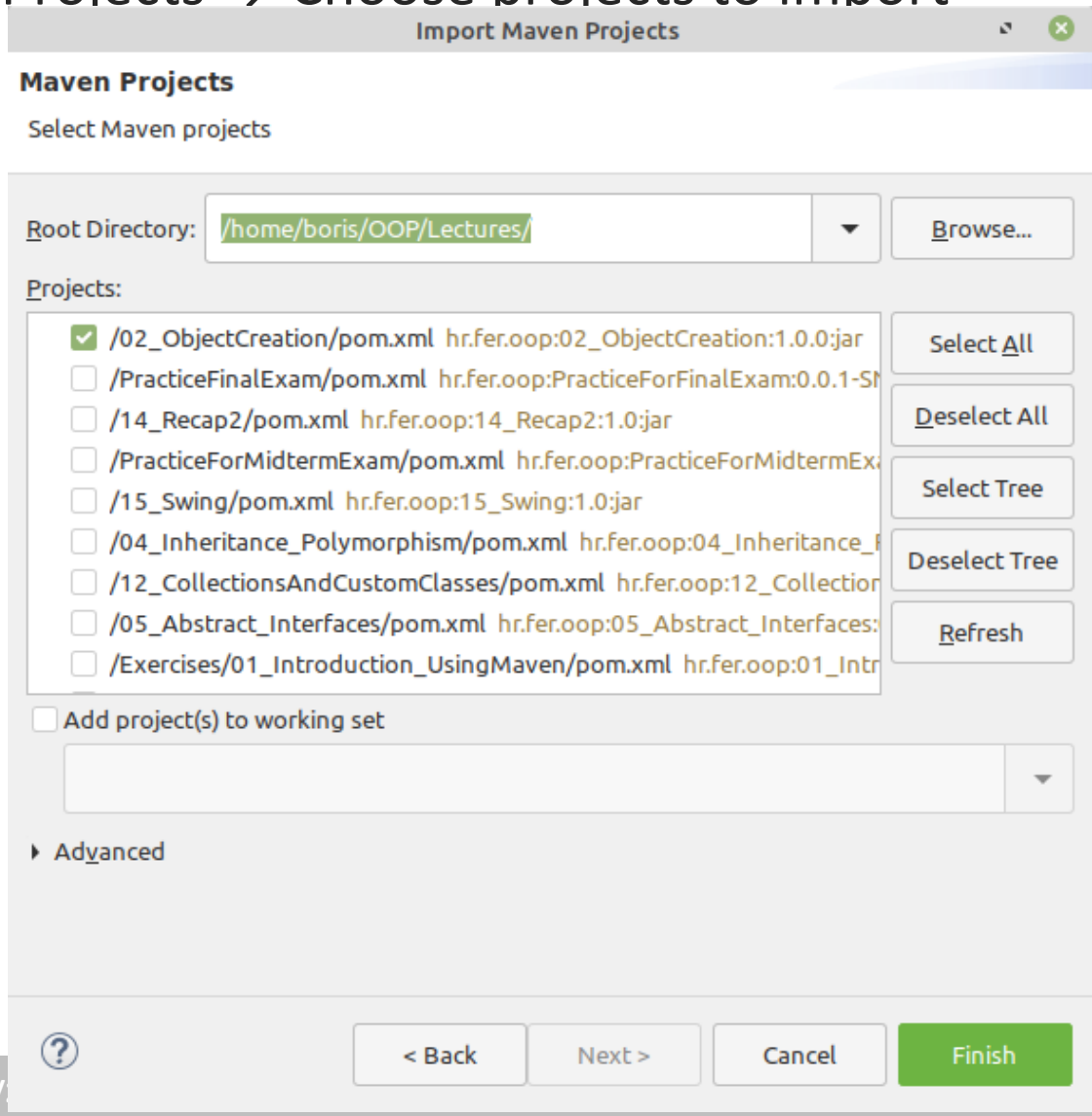
Struktura Maven projekta

- Vršna mapa projekta sadrži datoteku pom.xml koja postavlja:
  - identifikator projekta
    - važno ako se programske biblioteke smještaju na Mavenov centralni repozitorij
  - ovisnosti o programskim bibliotekama trećih strana
    - Maven ili IDE će ih automatski skinuti ako ne postoje lokalno
  - verziju Jave koju treba koristiti
  - …
- Maven/Gradle projekt (po konvenciji) koristi drugačiju strukturu mapa
  - *src* → *src/main/java*
    - src može sadržavati i ostale tipove datoteka (resurse, testove, …)
  - *bin* → *target/classes* (kod gradlea je *build/classes*)
    - *target* može sadržavati i ostale binarne datoteke u posebnim mapama, npr. target/test-classes, …

Uvoz Maven projekta u Eclipse

- Import → Existing Maven Projects → Choose projects to import

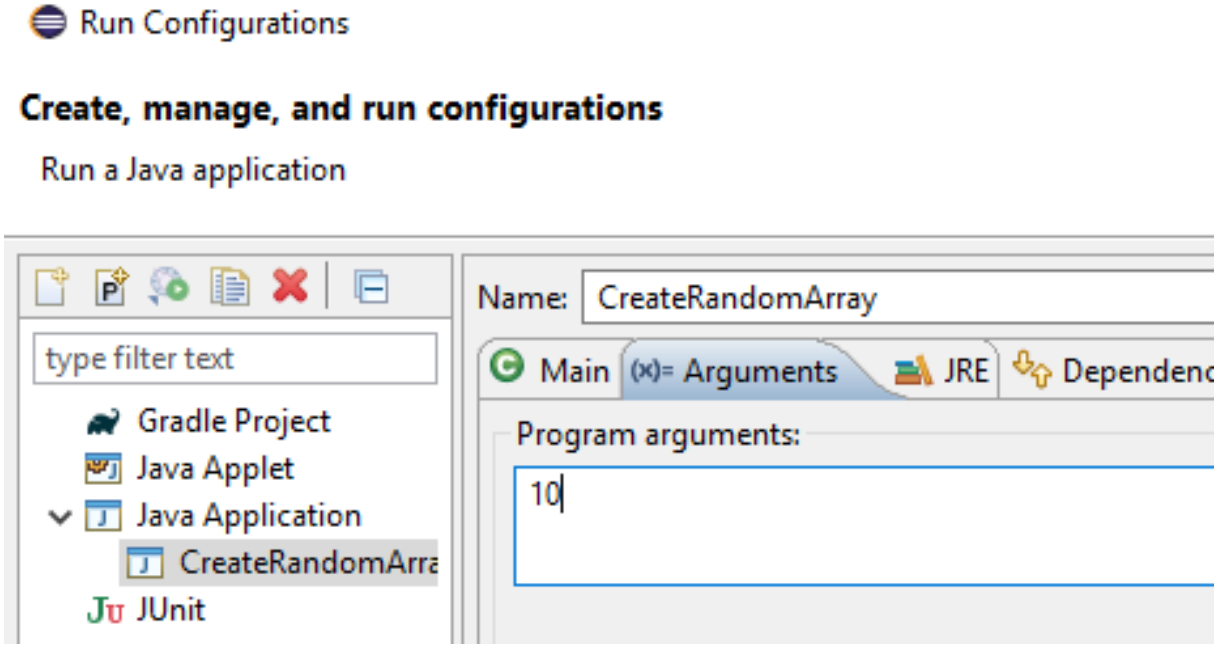




Kako postaviti argumente programa u Eclipseu

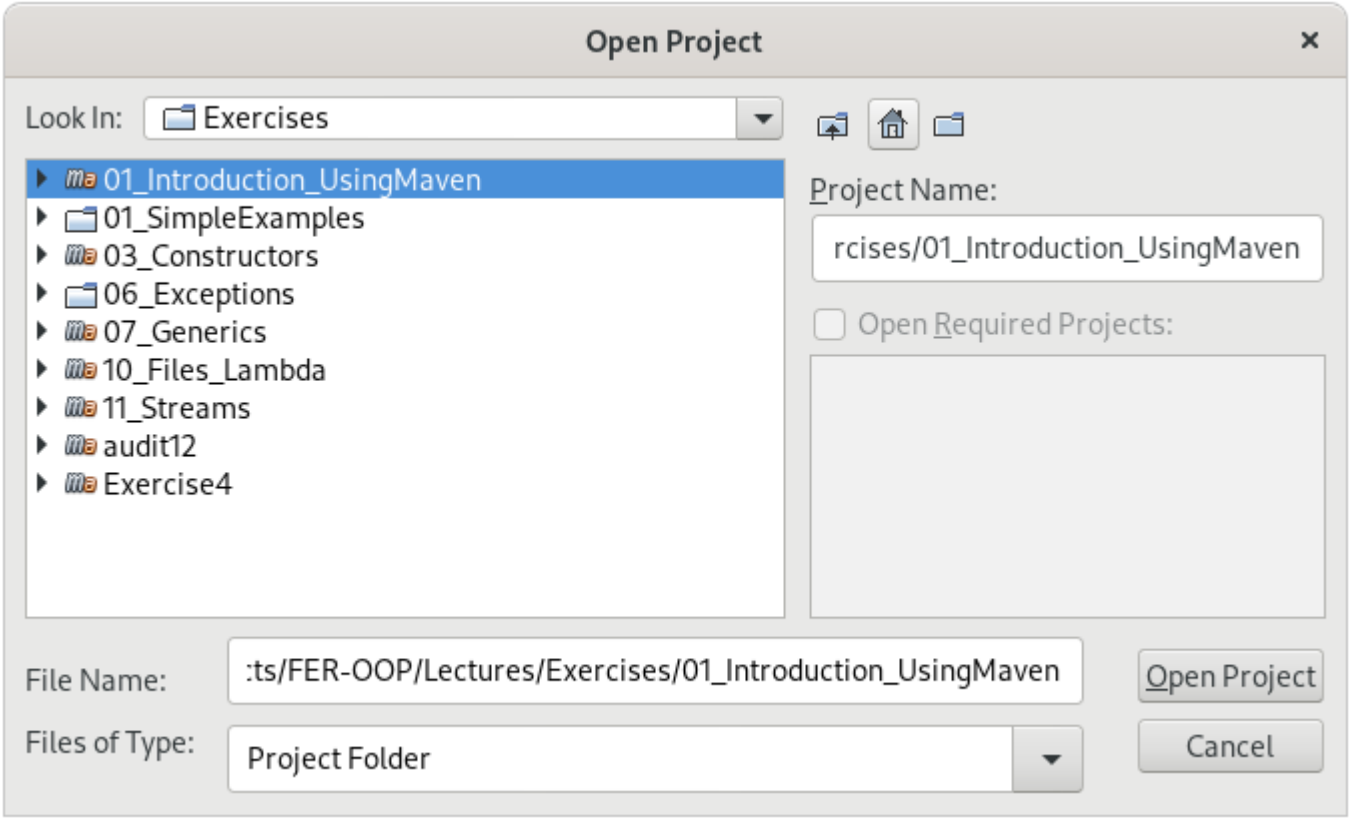
- Desni klik na datoteku s glavnim programom → Run As → Run Configurations

- Unijeti argumente u karticu *Arguments*



Uvoz Maven projekta u NetBeans

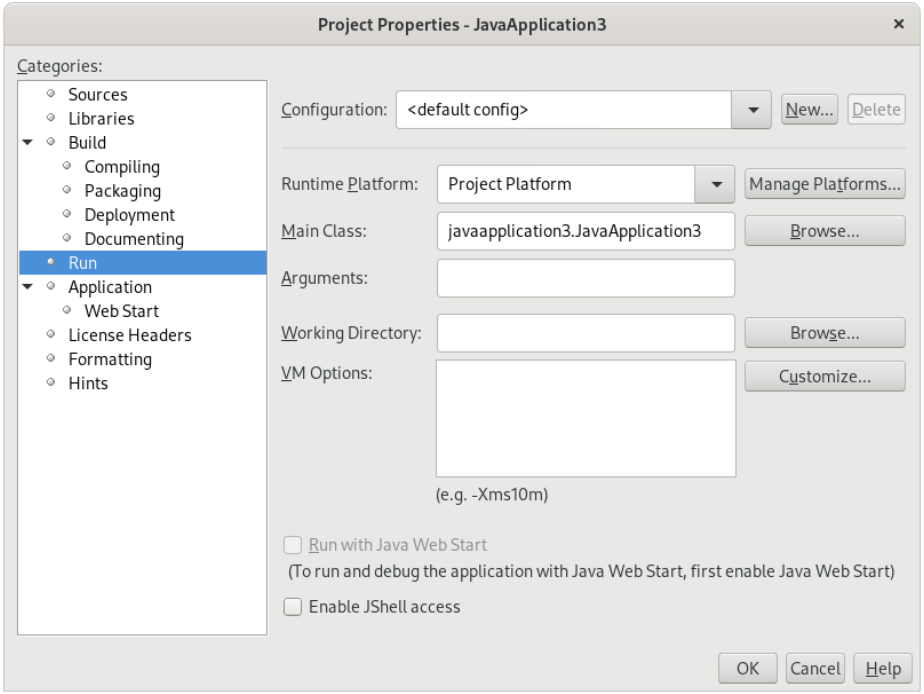
- File → Open Project



- U prozoru koji se otvori Maven projekti imaju ikonicu „ma”
- Odabrati projekt i kliknuti Open Project

Kako postaviti argumente programa u NetBeansu

- Desni klik na projekt → Set Configuration → Customize…



- Kliknuti na gumb Browse… koji se nalazi desno od polja Main Class te odabrati glavnu klasu
- U polje Arguments upisati argumente i kliknuti na gumb OK
- Desni klik na projekt → Run

Stog (engl. *stack*) i hrpa (engl. *heap*)

- Na stogu se pohranjuju lokalne varijable
  - deklaracija lokalne varijable nekog primitivnog tipa zauzima prostor odgovarajuće veličine na stogu i pohranjuje neku vrijednost unutra
- Omogućuje izvođenje funkcija i rekurzija
  - pozivanje metode stavlja argumente metode i povratnu adresu na stog – završetak metode „skida” elemente sa stoga
- Na hrpi se nalaze ostale vrste podataka
  - dinamički alocirani objekti
  - izvršni kod programa
  - informacije u klasama i metodama
  - konstante

Nizovi (polja) u Javi

- Nizovi se deklariraju s `tip[] naziv_varijable;`

- Deklaracija polja nije ujedno i stvaranje polja!
- Na stogu se stvara prostor (obično veličine 32 ili 64 bita) za deklariranu varijablu.
  - takva varijabla je referenca (slično pokazivaču) - sadržaj varijable će biti adresa kontinuiranog bloka na hrpi u kojem će biti smješteni elementi niza
  - kontinuirani blok na hrpi će biti kreiran ključnom riječi *new*
  - inicijalni sadržaj varijable je *null* što znači da trenutna referenca ne pokazuje na niti jedan objekt

Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- (Vrlo) pojednostavljeno stanje u memoriji nakon koraka 1 (za argument n=5)

Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

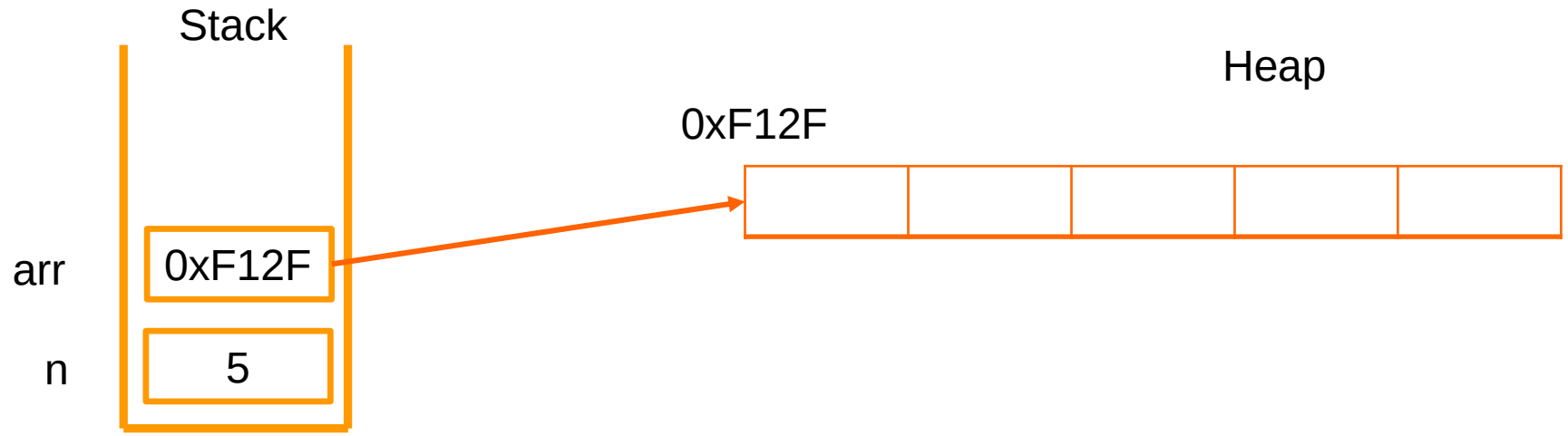
- Pojednostavljeno stanje u memoriji nakon koraka 2

Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- korak 3 – za ilustraciju, zamislimo da je niz kreiran na adresi 0xF12F

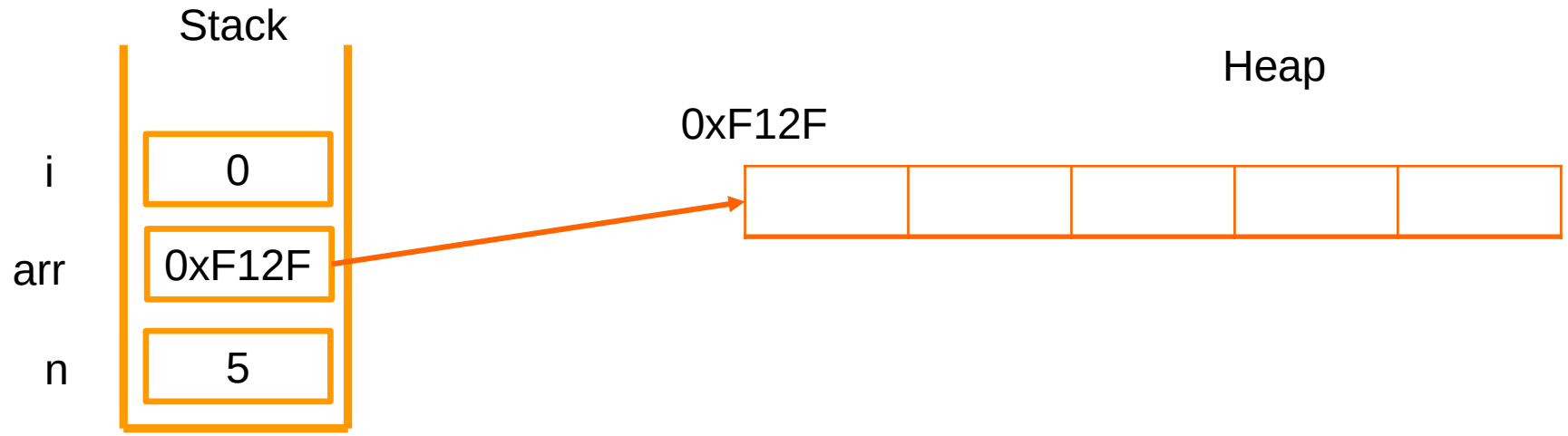


Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- korak 4 – inicijalni korak petlje

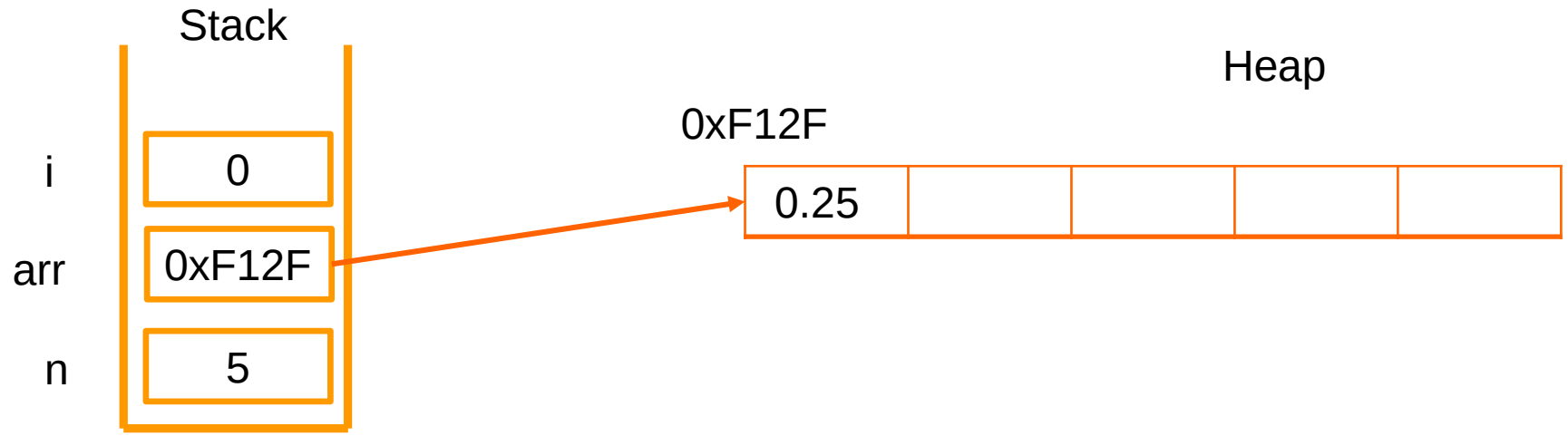


Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- korak 5 – za *i* = 0, a pseudo-slučajni broj 0,25

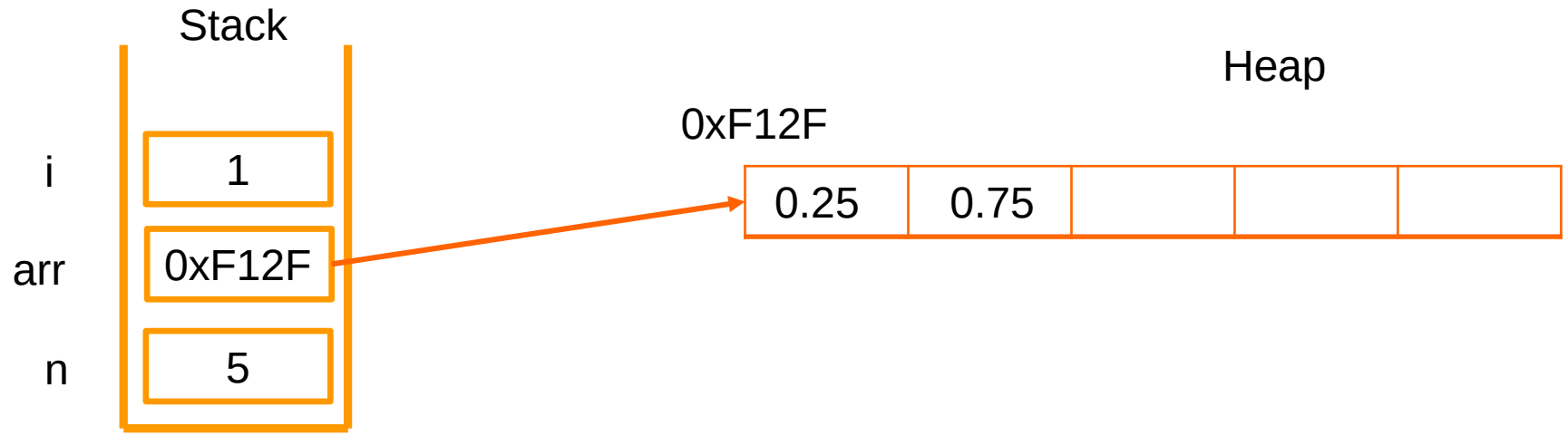


Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- korak 5 – za *i* = 1, i pseudo-slučajni broj 0,75

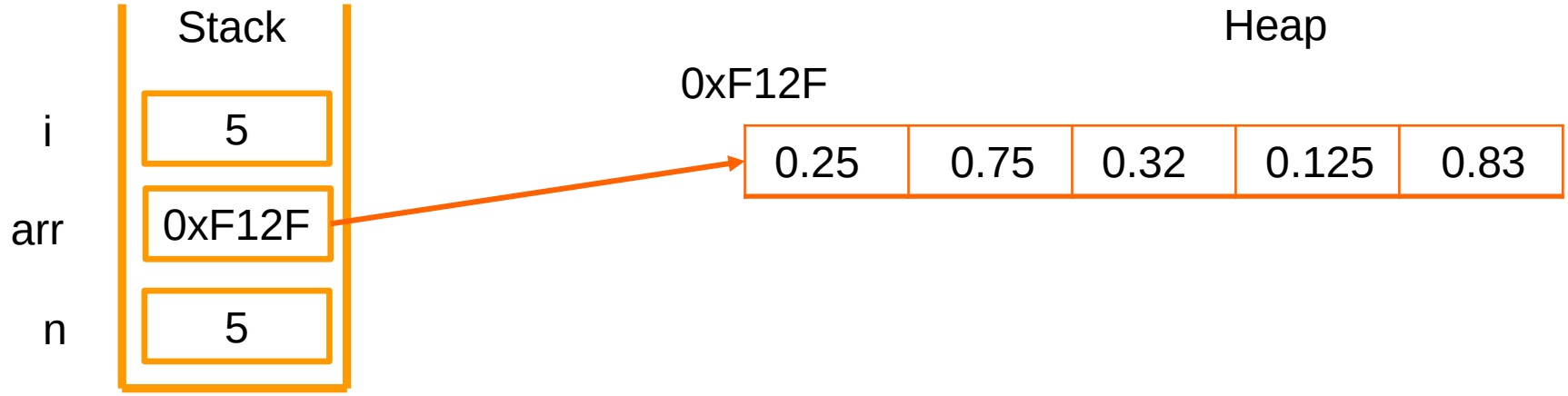


Stvaranje niza u Javi

```
public static double[] create(int n) { // 1
  double[] arr;                        // 2
  arr = new double[n];                 // 3
  for(int i=0; i<n; i++) {             // 4
       arr[i] = Math.random();         // 5
  }
  return arr;                          // 6
}
```

…02_ObjectCreation/src/main/java/hr/fer/oop/objectcreation/CreateRandomArray.java

- korak 6 – što `return arr` znači? → vraća se 0xF12F (referenca na polje koje počinje na toj adresi)

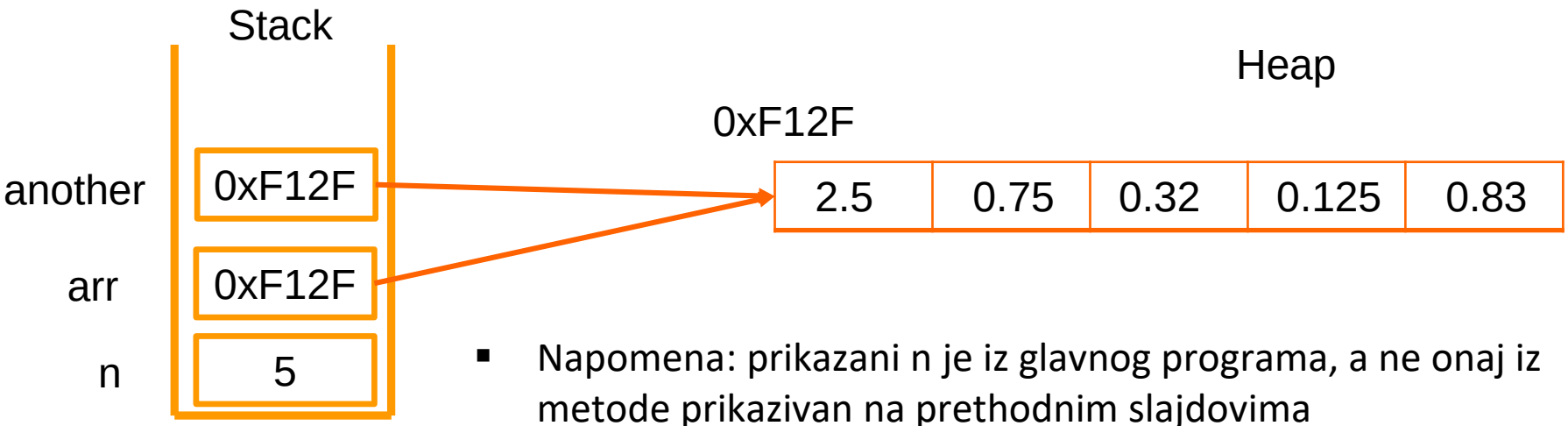


operator = kopira vrijednosti na stogu

- operator = kopira sadržaj na stogu
  - u ovom slučaju kopirana vrijednost predstavlja adresu

```
double[] arr = create(n);
double[] another = arr; //što se događa u ovom koraku?
another[0] = 2.5;
System.out.printf("#1 = %.4f %.4f %n", arr[0], another[0]);
```

…02_ObjectCreation/…/CreateRandomArray.java

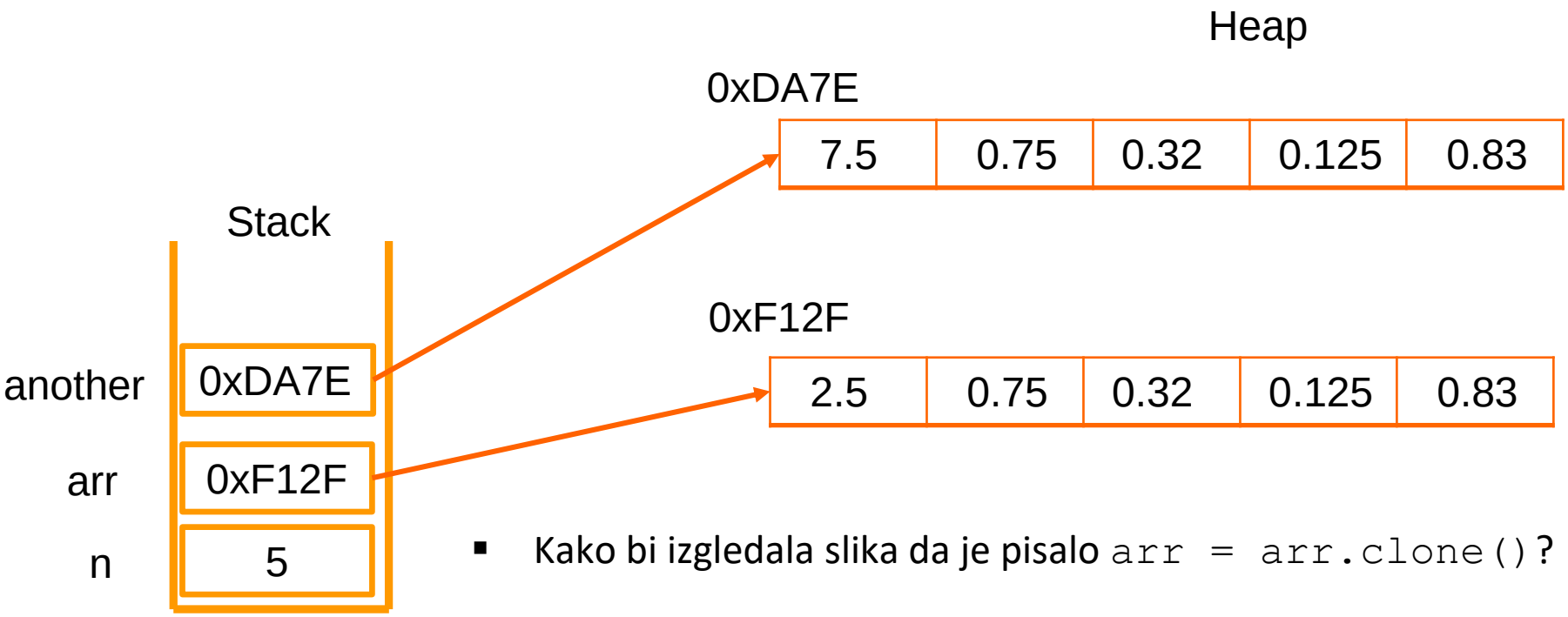


- Napomena: prikazani n je iz glavnog programa, a ne onaj iz metode prikazivan na prethodnim slajdovima

Kloniranje polja

- Metoda *clone* stvara novo polje i kopira elemente

```
another = arr.clone(); //što se događa u ovom koraku?
another[0] = 7.5;
System.out.printf("#2 = %.4f %.4f %n", arr[0], another[0]);
```

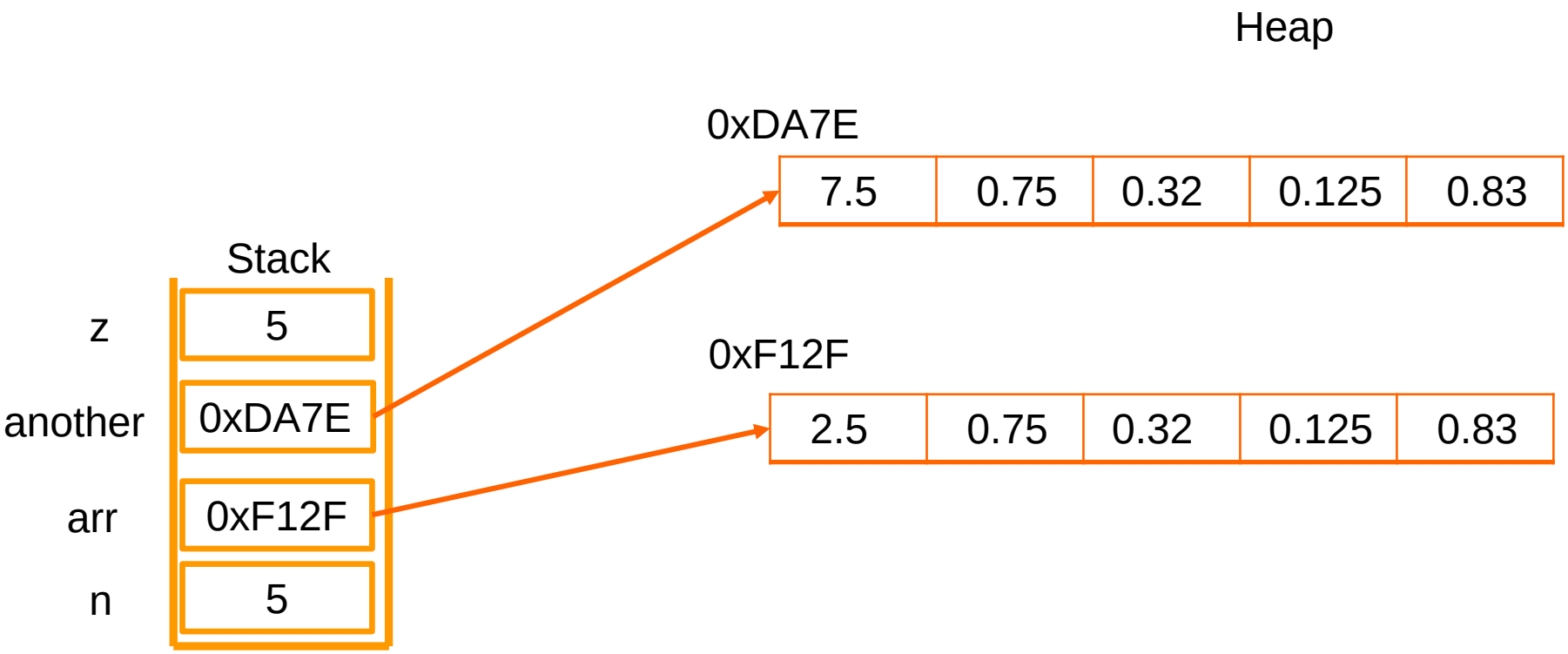


- Kako bi izgledala slika da je pisalo `arr = arr.clone()`?

operator = kopira vrijednosti na stogu

- operator = kopira vrijednosti na stogu

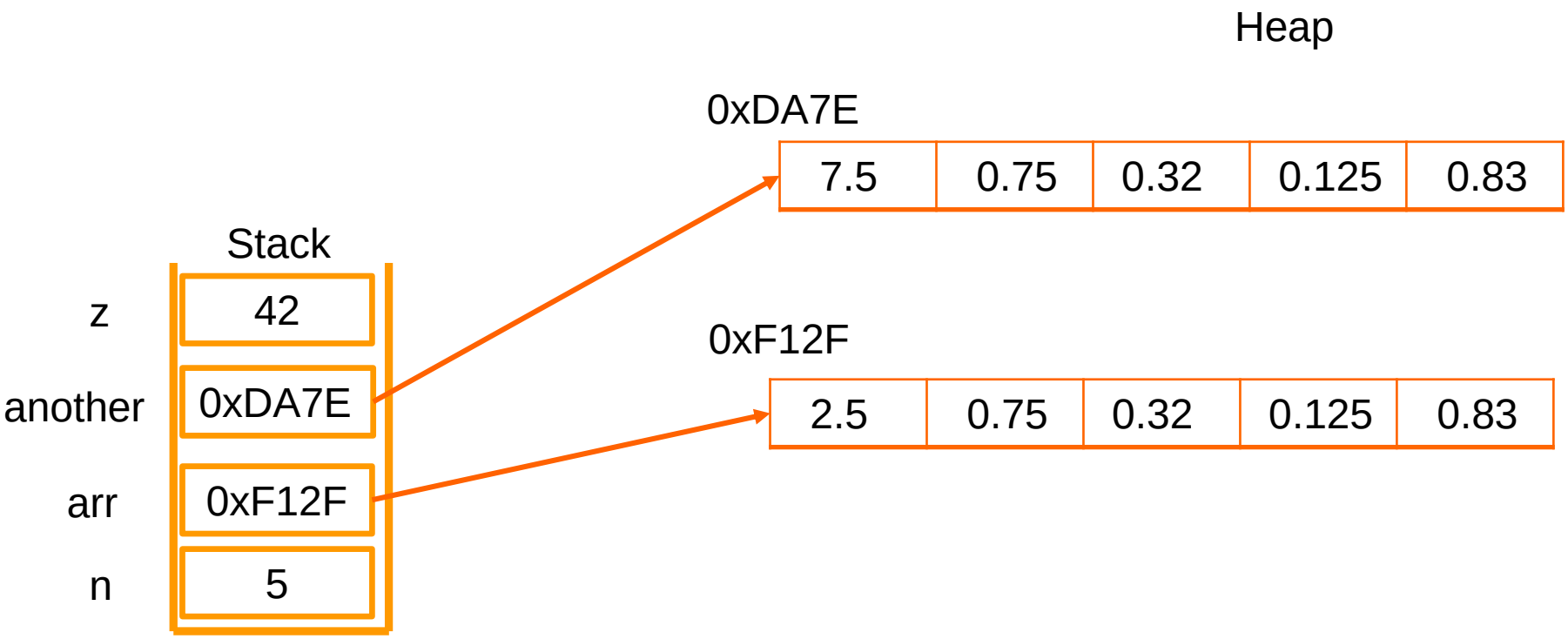
```
int z = n;
```



operator = kopira vrijednosti na stogu

- operator = kopira vrijednosti na stogu
  - u ovom slučaju kopirao je sadržaj koji predstavlja vrijednost varijable

```
z = 42;
System.out.printf("#3 = %d %d %n", n, z);
```



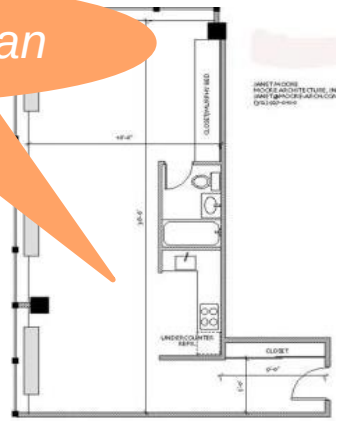
Klase i objekti

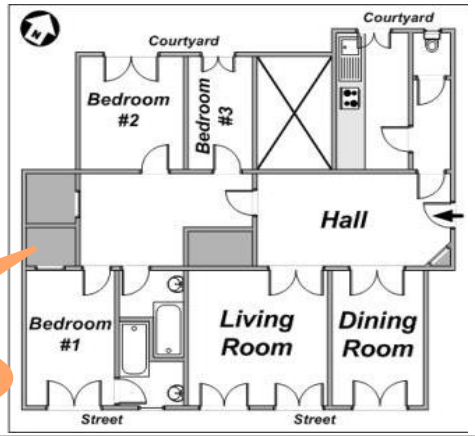
- Klasa definira kako će izgledati neki objekti nakon što se kreiraju
  - definira svojstva/sadržaje (varijable i metode) koje će objekti te klase posjedovati
- Objekt = 1 primjerak (instanca) klase

Objekti – instance klase *Veliki stan*

Objekti – instance klase *Mali stan*

Klasa: *Mali stan*



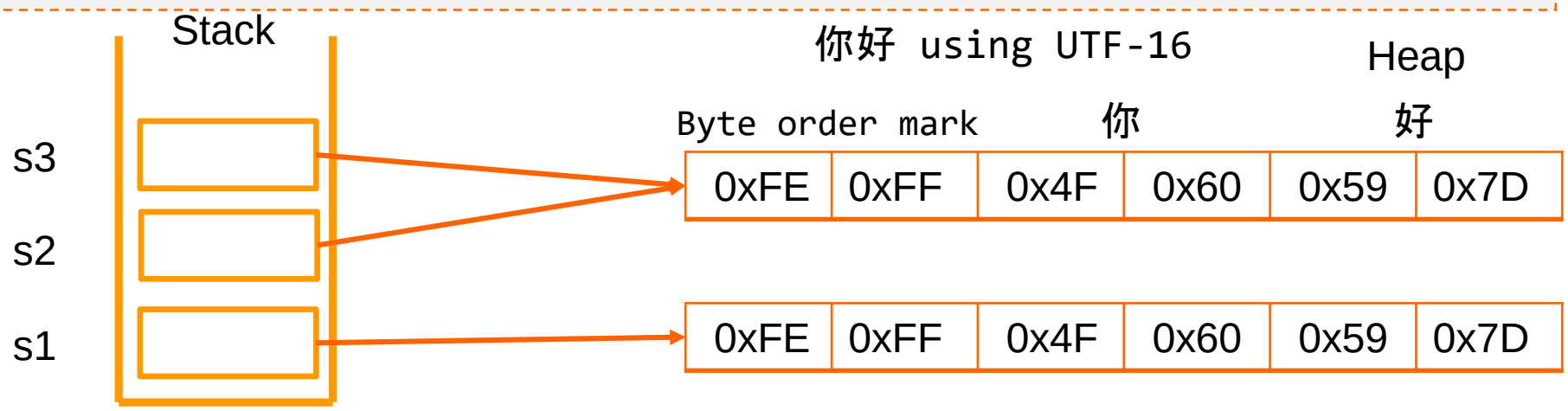


Klasa: *Veliki stan*

Stringovi u Javi

- String služi za pohranu i rad s nizom znakova
  - *char* je primitivni tip, *String* je klasa
  - instanca *Stringa* čuva znakove kao niz bajtova (sadržaj ovisi o načinu kodiranja znakova)
- Varijable definirane kao *String* su reference!
  - novi stringovi (objekti) se kreiraju operatorom *new*

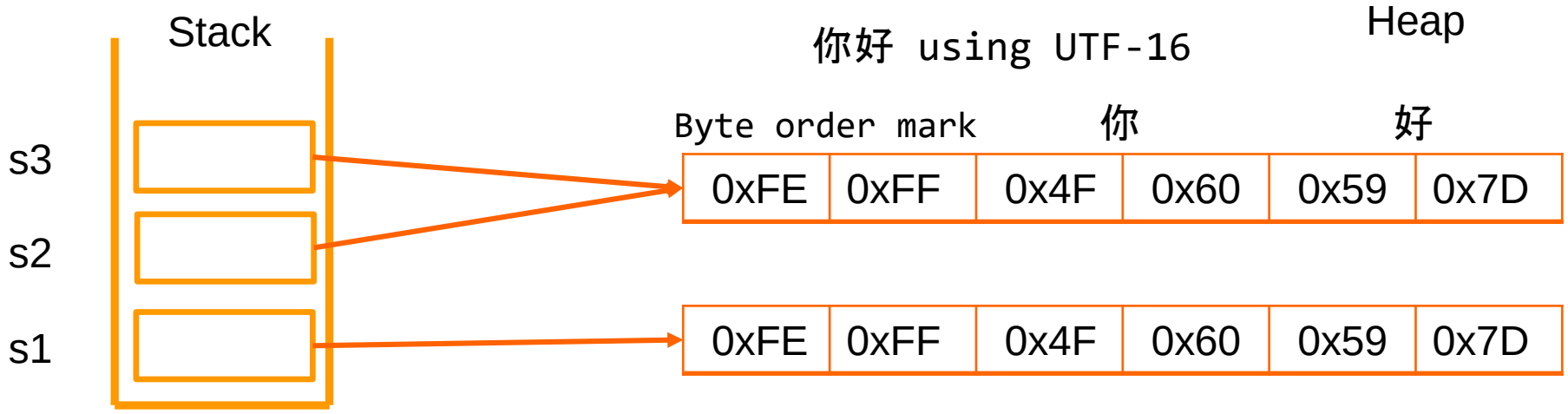
```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = new String("你好"); //Nǐ hǎo
String s3 = s2;
```



Usporedba stringova (1)

- Operator == uspoređuje vrijednosti na stogu!
  - *s2 == s3 → true*
  - s1 == s2 → *false*

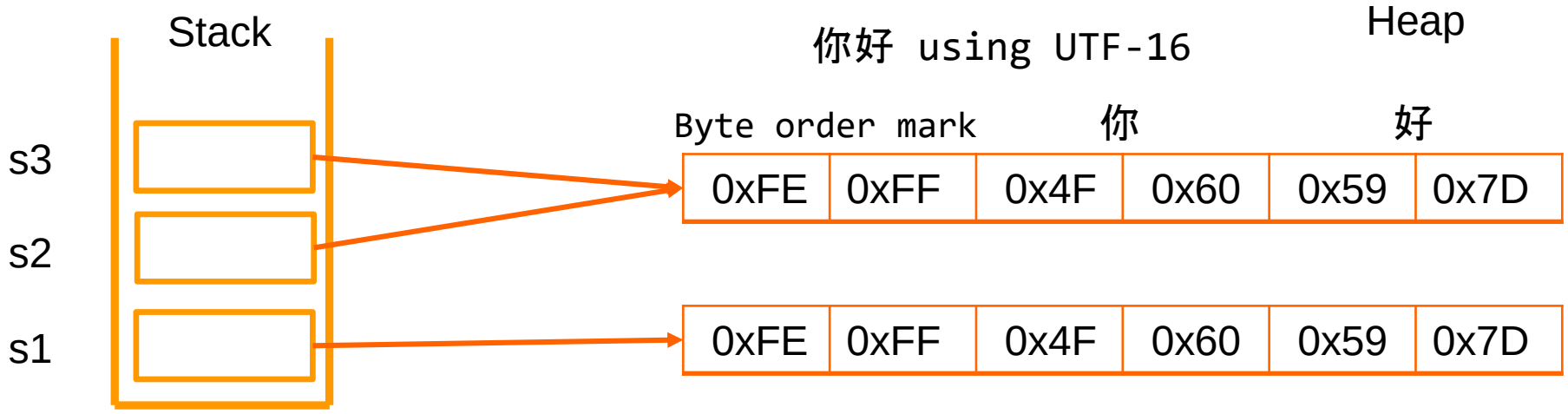
```
String s1 = new String("你好"); //Ni hǎo
String s2 = new String("你好"); //Ni hǎo
String s3 = s2;
```



Usporedba stringova (2)

- Kako usporediti sadržaj stringova ?
  - *s2.equals(s3)* → *true*
  - *s1.equals(s2)* → *true*

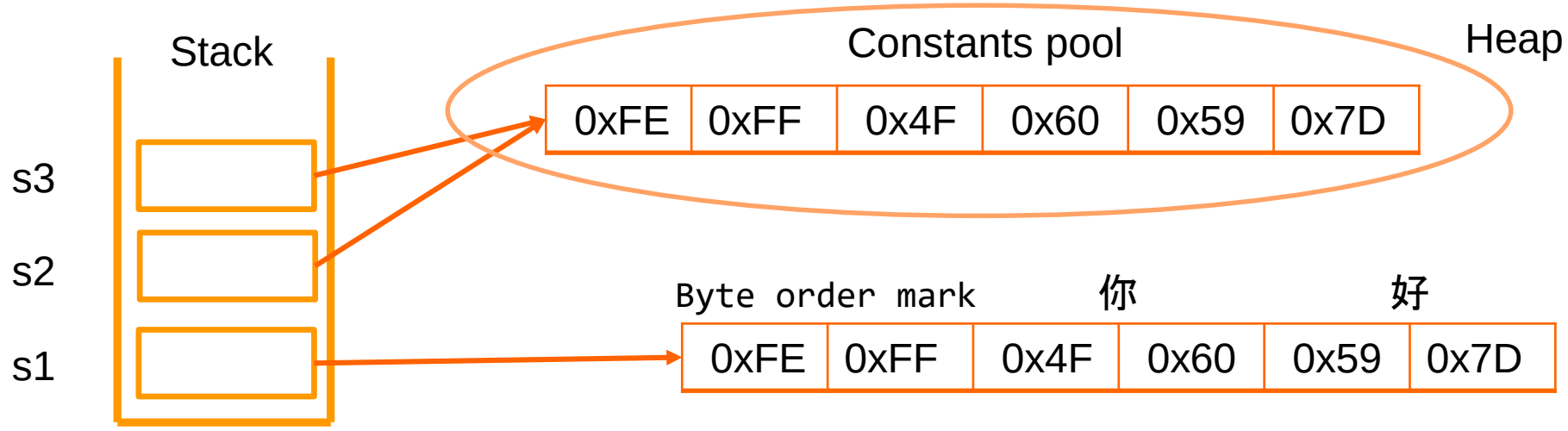
```
String s1 = new String("你好"); //Ni hǎo
String s2 = new String("你好"); //Ni hǎo
String s3 = s2;
```



Usporedba stringova (3)

- Napomena: sve stringovne konstante u Javi su smještene samo jednom u memoriji
  - samo *new* stvara novi string (može biti kopija postojećeg)
  - *s2 == s3* → *true* s1 == s2 → *false*
  - *s2.equals(s3)* → *true* *s1.equals(s2)* → *true*

```
String s1 = new String("你好"); //Nǐ hǎo
String s2 = "你好"; //Nǐ hǎo
String s3 = "你好";
```

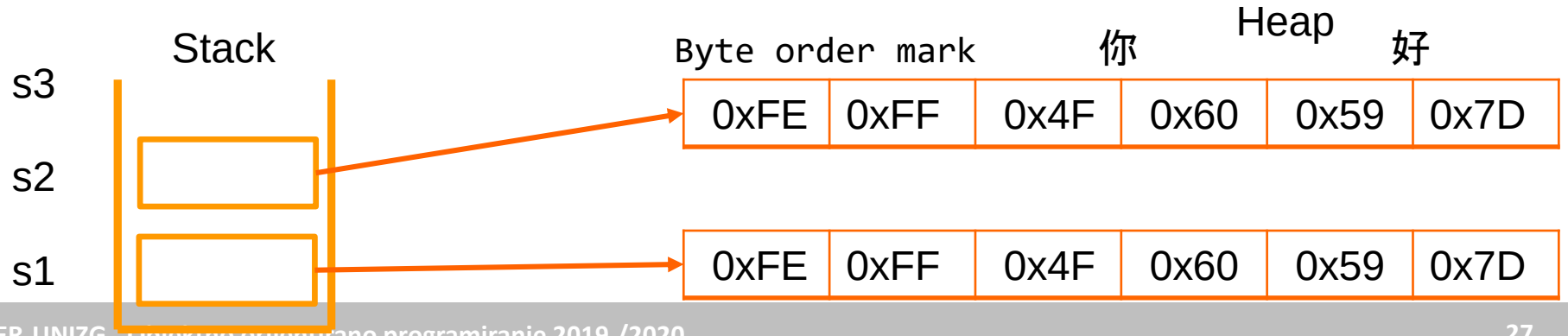


Metode objekata

```
String s1 = new String("你好");
char c = s1.charAt(0); // 你
String s2 = new String("你好");
boolean b = s2.equals(s1)
```

- Primijetite razliku u naredbama
  - *s2.equals(s3), s1.charAt(0)*

  u odnosu na neke naredbe (pozive metoda) iz prethodne prezentacije
  - *Math.abs, Integer.parseInt, Taylor.ePowerX*
- Methode *equals* i *charAt* iz klase *String* su metode objekta
  - potreban je odgovarajući objekt nad kojim će biti pozvane, tj. objekt čijem će sadržaju pristupati
- Statičke metode se mogu pozvati, a da ne postoji niti jedna instanca (objekt) te klase
  - o statičkim metodama više u sljedećim predavanjima)



Metode u klasi String

- Klasa *String* sadrži nekoliko korisnih metoda za rad sa stringovima
  - e.g. traženje podniza, zamjena dijela stringa, …
- String je nepromijenjiv!
  - sve methoda za manipulaciju stringom vraćaju novi string

```
String text = "The quick brown fox jumps over lazy dogs.";
String upperCase = text.toUpperCase();
System.out.println("Upper case text: " + upperCase);
System.out.println("Replacing fox with wolf: "
       + text.replace("fox", "wolf"));
int position = text.indexOf("quick");
System.out.println("quick starts at index: " + position);
System.out.println(text.substring(position, position + 15));
System.out.println("Original: " + text);
```

- Izvršite i analizirajte kod iz primjera te isprobajte neke druge metode iz klase String

…02_ObjectCreation/…/WorkingWithStrings.java

Spajanje stringova

- Operator + se može koristiti za spajanje stringova
  - može se koristiti u kombinaciji s brojevima ili drugim objektima
    - moguće zbog koncepta koji se naziva *autoboxing* i postojanja metode *toString* (bit će objašnjeno u nekom od narednih predavanja)

```
String text = "The quick " + "brown ";
text += "fox jumps over ";
text += 3;
text += " lazy dogs.";
System.out.println(text);
```

…02_ObjectCreation/…/WorkingWithStrings.java

- Napomena: ne zaboravite da su stringovi nepromjenjivi
  - u slučaju velikog broja spajanja preporuča se korištenje klase *StringBuilder*

Životni ciklus objekta

- Što se događa s objektima na hrpi?
  - stvara se eksplicitno operatorom *new*,
  - može nastati korištenjem drugih metoda, npr. novi stringovi nastaju metodama za manipulaciju stringovima
  - postoji li problem curenje memorije (engl. memory leak)?
- Nema razloga za (pretjeranu) brigu - *Java* ima sakupljač smeća (engl. *Garbage collector, GC)*
  - GC uklanja objekte koji više nisu referencirani
    - trenutak izvođenja GC-a nepoznat (nije vezan uz doseg varijable)
  - GC će prije uklanjanja objekta iz memorije nad njim izvršiti metodu *finalize* - u većini slučajeva nema potrebe za pisanjem ove metode
  - previše nepotrebno stvorenih objekata uzrokovat će češće pokretanje GC-a i potencijalno može usporiti izvršavanje programa
    - stoga je npr. StringBuilder primjereniji za česta spajanja stringova

Standardni ulaz/izlaz

…02_ObjectCreation/…/ReadFromStandardInput.java

- System.out.println, System.out.printf, System.err.println, …
  https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/Formatter.html#syntax
- Klasa Scanner za učitavanje vrijednosti s nekog ulaza
  - u donjem primjeru sa standardnog ulaza, ali primjenjivo i na druge izvore

```
Scanner sc = new Scanner(System.in);
System.out.println("Please enter an integer number");
int x = sc.nextInt();
System.out.println("Entered: " + x);
System.out.println("Now, enter a floating point number");
double y = sc.nextDouble();
System.out.println("Entered: " + y);
System.out.println("Enter several lines. Use Q or q to quit.");
while(sc.hasNextLine()) {
        String line = sc.nextLine();
        if (line.equalsIgnoreCase("Q")) break;
        System.out.println(line);
}
```