UVOD:

- Java programi (.java) -> prevođenje u bytecode (ekstenzija .class, bytecode nije izvršni kod kao C++) -> class loader premješta .class datoteke u RAM->Bytecode verifier provjerava ispravnost bytecoda te moguće kršenje sigurnosnih ograničenja -> JVM pokreće(interpretira) bytecode koji čita iz RAM-a (tijekom izvođenja i piše u RAM)

- Garbage collector, nema pokazivača, nasljeđivanje samo jedne klase, izvođenje na više platformi (Što je istinito/krivo za Javu?)

- JRE – okruženje samo za pokretanje Java aplikacija
- JDK – okruženje za razvoj i pokretanje Java aplikacija

Klase:

- predstavlja nacrt za kreiranje objekta

- [public] [abstract] [final] class ImeKlase [extends ImeNadklase] [implements ImeSučelja]{

```
        [konstruktor]
        . . .
        deklaracija varijabli
        . . .
        deklaracija metoda
        . . .
    }
```

- public class Date {

```
    int day;
    String month;
    int year;
}
```

Instanciranje objekta:

```
Date firstDate = new Date();
```

Pristupanje varijablama:

```
firstDate.day = 1;
firstDate.month = "Listopad";
firstDate.year = 2011;
```

Usporedbe:

```
if (f.day == s.day && f.month.equals(s.month) && f.year == s.year)
	return true;
else
	return false;
```

Čitanje podataka:

```
Scanner unos = new Scanner(System.in);
int uneseniBroj = unos.nextInt();
```

Ispis podataka:

```
System.out.println("Hello World!");
```

Getter and Setter(javne metode za upravljanje varijablama):

```
private int x;
	public int getX() {
	return x;
}
public void setX(int x) {
	this.x = x;
}
```

Konstruktori:

- metode za kreiranje objekata
- Ime kao i ime klase (kao i ime java datoteke)
- početna inicijalizacija varijabli, ukoliko je potrebna
- Ukoliko nije naveden, compiler generira defaultni(bez parametara i inicijalizacije)

- može postojati više konstruktora, no svaki treba imati različiti tipove/broj parametara (*method overloading)*

```
public class Student {
	private String jmbag;
	public Student(String jmbag) {
		this.jmbag = jmbag;
	}
}
```

Metode:

```
	public double lineLength() {
		return Math.sqrt(Math.pow(p2.getX() - p1.getX(), 2) + Math.pow(p2.getY() - p1.getY(),
2));
	}
```

Statičke metode/varijable

- Metode/varijable koje imaju svojstvo "izravnog" pozivanja iz same klase (ključna riječ **static** se navodi iza modifikatora vidljivosti)
- public static final int BROJ_ECTS_BODOVA = 30;
- public static void resetBrojacObjekata() {

```
		brojacObjekata = 0;
	}
```

- Primjer poziva("Student" predstavlja samu klasu, a ne objekt te klase):
  - Student.resetBrojacObjekata()

Nazivi:

- U Javi se koristi konvencija CamelCase
- Nazivi klase se pišu s velikim početnim slovom
- Nazivi metoda i varijabli se pišu s malim početnim slovom
- Nazivi konstanti (enum je također konstanta) se piše velikim tiskanim slovima

Tipovi:

- Primitivni – nije potrebno kreirati objekte, možemo ih prepoznati jer se pišu malim početnim slovom
- Referentni – vezani uz klase te je za njih potrebno kreirati objekte (veliko početno slovo)
- Primjer: int i Integer, boolean i Boolean

OOP

Značajke:

- Generalizacija – određivanje zajedničkih svojstava
- **Nasljeđivanje** – prenošenje svojstava iz jedne u drugu klasu
  - nasljeđuje se samo jedna klasa
  - ukoliko je klasa "final" ne može se nasljediti
  - nasljeđena metoda se može nadjačati (method overriding)
  - sintaksa:
    - *public class Pas extends Zivotinja*
- **Enkapsulacija** – "skrivanje" podataka o konkretnoj implementaciji
  - koriste se javne metode za pristup varijablama (getteri i setteri)
- **Polimorfizam**
  - dinamičko pozivanje metoda klasa koje imaju zajedničku nadklasu ili sučelje (drugačije ponašanje ovisno o tipu objekta nad kojim se izvršava)
  - Primjer: 2 klase implementiraju isto sučelje sa definiranom metodom– u ovisnosti o tome na kojem objektu se poziva, drugačije se ponaša, tj. ponaša se na način na koji je definirano u metodi tog objekta

Apstraktne klase

- predviđene isključivo za nasljeđivanje
- iz njih se ne može kreirati objekt
- mogu (ne moraju) sadržavati apstraktne i "normalne" metode
- svaka klasa koja ima bar jednu apstraktnu metodu mora biti apstraktna

- apstraktna metoda ima samo "specifikaciju" metode (kao i interface)
- sintaksa: *public abstract class Zivotinja*

- Sve klase koje nasljeđuju apstraktnu nadklasu moraju implementirati sve apstraktne metode iz nadklase, ili trebaju i one biti apstraktne

Sučelja

- sadrže samo "specifikaciju" za klase koje ih implementiraju
- Klase mogu implementirati više sučelja (razlika u usporedbi s nasljeđivanjem)
- sintaksa:

```
public interface Mesojed {
        public void uhvatiPlijen(Zivotinja plijen);
}
```

- implementacija u klasi:

```
        public class Macka implements Mesojed {
                ...
                public void uhvatiPlijen(Mis hrana) {
                        this.hrana=hrana;
                }
                ...
        }
```

Final:

- sprečava nasljeđivanje(kod klasa) ili daljnje nadjačavanje (kod metoda)
- Sintaksa:
  - *public final class PerzijskaMacka {...}*
  - public final void predi() {System.out.println("Mrrrrr!");}

Super:

- pristupanje nekoj metodi ili varijabli iz nadklase
- poziv: super.**jediHranu()**;

Modifikatori vidljivosti:

	public	Bez modifikatora ("package")	protected	private
komponenta	Pristup iz svih ostalih klasa	Vidljivost unutar istog paketa	Vidljivost u podklasama	vidljivost iz klase
klasa	U svim klasama	Unutar paketa	-	- (osim za klasu unutar klase)
metoda	Vidljivost u svim ostalim klasama	-	Isti paket i podklase	vidljivost iz klase
varijable	Izravno korištenje	Unutar paketa	Isti paket i podklase	vidljivost iz klase

- **Synchronized** - omogućava sigurno izvođenje metoda u višenitnim sustavima
- **native -** implementacija metode je napisana u nekom drugom jeziku (npr. C-u) i predana je preko vanjskog resursa
- **abstract**
- **final**
- **extends**
- **implements**

Cast operacija

```
int suma = 100;    int broj = 20;
float prosjek = (float) suma / broj;
```

instanceof – provjera da li je neki objekt instanca neke klase

```
if(object instanceof Knjiga){
        ...
}else{
        ...
}
```

Iznimke

- način signaliziranja pogrešaka
- podaci u slučaju iznimke: - gdje se dogodila, koja je klasa iznimke, dodatne informacije o uzroku iznimke, naziv metode i broja linije gdje se dogodila
  - ```
    Exception in thread "main" java.lang.NullPointerException
        at test.ExceptionTest.main(ExceptionTest.java:13)
    ```
- Klase iznimaka: Throwable(Klasa koju sve iznimke nasljeđuju), Error(pogreške unutar JVM-a), Exception(Neispravan URL, otvaranje datoteke koja ne postoji, **MORA** biti obrađena), RuntimeException(pogreške u programiranju - ClassCastException, NullPointerException, AritmeticException)
- ```
  try {
        System.out.print("Unesite brojeve : ");
        int prviBroj = unos.nextInt();
        int drugiBroj = unos.nextInt();
        System.out.println("Rezultat " + prviBroj + "/" + drugiBroj
        + " = " + prviBroj / drugiBroj);
  } catch(ArithmeticException ex) {
        System.out.println("Dogodila se aritmetička pogreška" +ex.getMessage());
        System.out.println("Radi se o sljedecoj iznimci: ");
        ex.printStackTrace();
  } finally {
        programski kod koji se obavlja u svakom slucaju
  ```

```
}
```

- ```
  public static void metoda(int i) throws InvalidPasswordException,
        IllegalDataException, DatabaseUnavailableException
  ```

- ```
  throw new RuntimeException("Neka poruka!");
  ```
- Try blok sa resursima:

```
try(FileReader citac = new FileReader(datoteka)) {
      char znak = (char) citac.read();
      System.out.println("Pročitani znak je " + znak);
}
catch (FileNotFoundException ex) {
      ex.printStackTrace();
}
```

Javadoc:

- @author (koristi se kod klasa i sučelja, obvezan)
- @version (koristi se kod klasa i sučelja, obvezan)
- @params (metode i konstruktori)
- @return (metode)
- @exception (kod bacanja iznimaka – sinonim @throws)
- @see (koristi se kod referenciranja na druge izvore)
- @since (označava inačicu otkad je nešto uvedeno)

- @serial (specifikacija serijalizacije - ili @serialField ili
- @serialData
- @deprecated (oznaka zastarjelosti)
- Kod elemenata programskog koda potrebno je koristiti tag *<code></code>*

Dinamičke strukture:

Polja – Objekti koji mogu sadržavati i primitivne tipove i objekte

- ```
  String[] pozdravi = new String[3];
  pozdravi[0] = "Dobro jutro!";
  pozdravi[1] = "Dobar dan!";
  ```
- ```
  int primBrojevi[] = new int[] { 2, 3, 5, 7, 11, 13, 17};
  ```
- ```
  int[] array = new int[100];
  ```
  ```

Zbirke

Liste(ArrayLIst,LinkedList, Vector) – čuva poredak i ne provjerava duplikate

- ```
  List<String> novaLista = new ArrayList<String>();
  novaLista.add("Dobar dan!");
  for (String pozdrav : novaLista) {
        System.out.println(pozdrav);
  }
  ```

- ```
  for (Iterator<String> it = novaLista.iterator(); it.hasNext();) {
              String value = it.next();
   }
  ```

- Klasa Collections – statičke metode kojima se manipulira elementima unutar zbirke
  - sort, reverse, min, max, addAll, binarySearch
  - `Collections.sort(listaOsoba, new DatumRodjenjaComparator());`

Setovi(HashSet, TreeSet, LinkedHashSet) – unikatni članovi

- `Set<String> mySet = new HashSet<String>();`
- `SortedSet<Zupanija> setZupanija = new TreeSet<Zupanija>(new GustocaNaseljenostiComparator());`

Mape(HashMap, Hashtable, TreeMap) – parovi ključ vrijednost, unikatni ključevi

- Map<Long, String> novaMapa = new HashMap<Long, String>();
- novaMapa.put(new Long(10), "vrijednost");
- Set<Long> setKljuceva = novaMapa.keySet();
- Set<String> setVrijednosti = novaMapa.values();
- Ako se u mapu doda element čiji ključ već postoji, stari par „ključ-vrijednost“ se zamjenjuje novim parom

Sortiranje

```
public class StudentComparator implements Comparator<Student> {

	public int compare(Student st1, Student st2) {
		if (st1.getBrojBodova() > st2.getBrojBodova()) {
			return 1;
		}
		else if (st1.getBrojBodova() < st2.getBrojBodova()) {
			return -1;
		}
		else {
			return 0;
		}
	}
}
```

```
class StudentComparator implements Comparator<Student> {

   @Override
   public int compare(Student st1, Student st2) {
      if (st1.getScore().compareTo(st2.getScore())<0) {
         return 1;
      } else if (st1.getScore().compareTo(st2.getScore())>0) {
         return -1;
      } else {
         return 0;
      }
   }
}
```

OBJAŠNJENJE:

Brojka 1 unutar compare metode mijenja poredak. Dakle nju treba potražiti i odredimo si 2 vrijednosti (npr 4 i 5)

- a) Unutar prvog slučaja, kod broja 1 imamo uvijet `st1.getBrojBodova() > st2.getBrojBodova()` - dakle ukoliko je 4>5 zamijeni poredak. S obzirom da to nije točno, poredak ostaje kako je i zaključujemo da ovaj komparator sortira uzlazno(npr 4,5,6,...)
- b)Unutar drugog slučaja trebamo znati što vraća metoda *compareTo(Object arg1).* Dakle ona vraća -1 ukoliko je prva vrijednost manja od druge, 0 ukoliko su jednake i 1 ukoliko je prva vrijednost veća od druge. Opet si odredimo 2 broja (4 i 5), tražimo kad metoda vraća broj 1 – te s obzirom da je 4 zaista manje od 5, mjenjamo poredak te iz toga dobijemo da komparator sortira silazno (npr. 5,4,3,2..)

Enumeracije

- konačan skup vrijednosti koje se definiraju kao varijable klase
- Implicitno označene modifikatorima static i final
-
```
public enum Jezik {
	HRVATSKI(1), ENGLESKI(2), NJEMACKI(3),
	FRANCUSKI(4), TALIJANSKI(5), RUSKI(6), KINESKI(7);

	private int kod;
	private Jezik(int kod) {
		this.kod = kod;
	}
	public int getKod() {
		return kod;
	}
}
```

-
```
for (Jezik jezik : Jezik.values()) {
	System.out.println("Jezik : " + jezik);
}
```

Literatura:

Predavanja iz Jave, 2014.(https://moj.tvz.hr/index.php?pred=19321),
Pred. Aleksander Radovan , dipl. ing.