קורס JAVA

מרצה: עסאקלה שאדי

הרצאה מספר 3

מחלקה – לתיאור עצם

- מחלקה היא טיפוס נתונים בעלת תכונות ופעולות
  - תכונות הן משתנים
  - פעולות הן פונקציות (מתודות) שניתן להפעיל על טיפוס הנתונים
- לדוגמא:
  - תאריך מאופיין ע"י ימים, חודשים ושנה
  - על תאריך ניתן לבצע פעולה של "שינוי" שבו משנים ימים, חודשים או שנים
  - על תאריך ניתן לבצע פעולה של הצגה אשר מציגה את התאריך, או חלק ממנו.

Date
+ dd: int + mm: int + yy: int
+ change() : void + show() : void

מימוש המחלקה Date

Date
+ dd: int + mm: int + yy: int
+ change() : void + show() : void

```
public class Date{
        public int dd;
        public int mm;
         public int yy;

        public void change() {
                dd++;
                 mm++;
                }
        public void show(){

        System.out.print(dd+"/"+mm);
                System.out.print("/"+yy);

                }
}
```

יצירת אובייקט למחלקה

- כדי ליצור במחלקה הראשית אובייקט של המחלקה נרשום את הפקודה:
- className objName= new className();

- new פקודה מיוחדת אשר בונה אובייקט.
- ערכי התכונות של האובייקט מאותחלים לאחר הקצאה בערכי ברירת מחדל לפי טיפוס הנתונים שלהם.
- בכל מחלקה מוגדרות פונקציות אתחול (אחת לפחות) ששמן כשם המחלקה. פונקציות אלו נקראות **בנאים**
- אופרטור ה- **new** מקצה זיכרון לאובייקט החדש ומאתחל אותו ע"י קריאה לבנאי

שימוש באובייקט

- כדי לפנות לתכונות משתמשים בסימן . (נקודה). למשל פעולת השמה:
- objName.var1 = value;
- כדי להפעיל פעולה (פונקציה):
- objName.funcName();

הפעלת עיקרון הכימוס

- הכימוס בגדול הוא הצורך להפוך את המחלקה לקופסא שחורה. הגישה לתכונות המחלקה צריכה להיות מבוקרת.
- לצורך גישה לתכונות המחלקה יש שלוש הרשאות:
- public, ניתן לגישה מכל מחלקה.
- private, גישה מהפונקציות של אותה מחלקה בלבד
- protected, יתנים לגישה מפונקציות של אותה המחלקה בלבד ומהמחלקות היורשות ממנה. (על ההורשה יורחב בהמשך).

הפעלת עיקרון הכימוס – סגירת גישה ושימוש ב set/get

```
public class Main {
        public static void
main(String[] args) {
        Date d1=new Date();
        d1.setDay(10);
        d1.setMonth(10);
        d1.setYear(2018);
        d1.show();
        }
}
```

```
public class Date {
private int dd;
private int mm;
private int yy;
public void setDay(int day)
{
        dd=day;
}
public void setMonth(int month)
{
        mm=month;
}
public void setYear(int year)
{
        yy=year;
}
public void show()

{
System.out.println(dd+"/"+mm+"/"+yy);
}}
```

הפעלת עיקרון הכימוס – סגירת גישה ושימוש ב set/get

```
public class Main {
        public static void
main(String[] args) {
        Date d1=new Date();
        d1.setDay(10);
        d1.setMonth(10);
        d1.setYear(2018);
   System.out.println(d1.getDay());
        d1.show();
        }
}
```

```
public int getDay()
}
return dd;
{
public int getMonth()
}
return mm;
{
public int getYear()
}
return yy;
```

ידפיס 10

תרגול

- בנה תרשים מחלקה שמתאר מכונית עם התכונות הבאות:

1. צבע.
2. מספר רישוי (מחרוזת).
3. שנת ייצור – שלם.

ועם הפונקציות הבאות:

1. פונקציות ציבוריות של set/get
2. פונקציה שמדפיסה את פרטי המכונית
3. פונקציה שמדפיסה את שנת הייצור של המכונית.

- בנה מערך שמכיל 3 מכוניות וקלוט ערכים עבור כל מכונית, הדפס את פרטי המכוניות על המסך בשימוש בשתי פונקציות ההדפסה לכל מכונית בנפרד.

המצביע this

- כאשר משתמשים בשמות משמעותיים בדרך כלל יוצרים משתנה מקומי בעל שם זהה לתכונה במחלקה (משתנה מחלקה).
- משתמשים במצביע this עבור משתנה המחלקה, כדי להבדיל בין משתנה מקומי למשתנה מחלקה.
- this הינו מצביע המייצג התייחסות לאובייקט הנוכחי שעליו עובדים.
- כדי לבצע השמה של פרמטר במשתנה מחלקה עם שמות זהים הסינטקס הוא:

```
this.תכונה = שם_פרמטר;
```

תיקון למחלקה Date

```
public class Main {
        public static void
main(String[] args) {
        Date d1=new Date();
        d1.setDay(10);
        d1.setMonth(10);
        d1.setYear(2018);
        d1.show();
        }
}
```

```
public class Date {
private int dd;
private int mm;
private int yy;
public void setDay(int dd)
{
        this.dd=dd;
}
public void setMonth(int mm)
{
        this.mm=mm;
}
public void setYear(int yy)
{
        this.yy=yy;
}
public void show()

{
System.out.println(dd+"/"+mm+"/"+yy);
}}
```

תרגיל

- בנה מערך מסוג Date, הכנס ערכים.
- הדפס את המערך על המסך בעזרת show()

```
public class Program {
	public static void main(String[] args) {
		Date[] dateArray;
		dateArray = new Date[5];
		int i;
		for(i=0;i<5;i++)
			dateArray[i] = new Date();
		for(i=0;i<5;i++)
		{
			dateArray[i].setDay(20+i);
			dateArray[i].setMonth(4+i);
			dateArray[i].setYear(2014+i);
		}
		for(i=0;i<5;i++)
			dateArray[i].show();
	}
}
```

methods overloading - העמסה

- הגדרת שתי פונקציות או יותר באותה מחלקה, עם אותו השם.
- הפונקציות צריכות להיות שונות בטיפוס ו\או מספר הארגומנטים.
- דוגמא: println
- יתרון: נוחות, תאימות אחורה בפונקציות.
- למשל, כדי לממש שתי פונקציות שמחזירות את המספר הגדול מבין השניים, אבל הפרמטרים מסוגים שונים, ללא העמסה הצטרכנו לעשות שתי פונקציות (בדומה לשפת C):
- public static int maxInt(int a, int b){}
- public static double maxDouble(double a, double b){}
- שמות מלאכותיים

methods overloading - העמסה

- פתרון בעזרת ההעמסה יתבצע דרך בניית פונקציות עם אותן שמות:

- public static int maxNum(int a, int b){}
- public static double maxNum(double a, double b){}
- עבור כל קריאה לפונקציה על פי טיפוסי הארגומנטים של הקריאה, המהדר ינסה למצוא את הגרסה **המתאימה ביותר** .
- כאשר אין התאמה מדויקת לאף אחת מחתימות הפונקציות הקיימות, המהדר מנסה לעשות המרות (casting) שאינן **מאבדות מידע**.
- אם לא נמצאת התאמה או שנמצאות שתי התאמות "באותה רמת סבירות" או שפרמטרים שונים מתאימים לפונקציות שונות המהדר מודיע על אי בהירות (ambiguity).

העמסה – פתרונות לבעיות

- שימוש בארגומנט אחד יותר מהאחרים, אז בונים פונקציה לטיפול רק בארגומנט ולא יהיה צורך לשלוח את כל השאר.
- שינויים עתידיים, למשל שימוש בארגומנט אחד, בפונקציה מסוימת, והתעורר צורך לבצע את אותן פעולות אבל עם בסיסים שונים:

```
public static int doSomething(int x, int base){
 // do something with x...
}

public static int doSomething(int x){
  // do something with x
}
```

תאימות לאחור

- המקרה הקודם במקום **להחליף** את חתימת השרות **נוסיף** פונקציה חדשה כגרסה מועמסת – תאימות לאחור.
  - יתרון - משתמשי הפונקציה המקורית לא נפגעים.
  - יתרון - משתמשים חדשים יכולים לבחור האם לקרוא לפונקציה המקורית או לגרסה החדשה ע"י העברת מספר ארגומנטים מתאים.

  - חסרון – שכפול של קוד
  - שכפול של קוד הוא דבר נורא בעולם התכנות...

תאימות לאחור פתרונות

- שכפול של קוד הוא דבר נורא בעולם התכנות...
- מוצאים מימוש של הפונקציה הישנה על ידי הפונקציה החדשה.

דוגמא:

```
public static int doSomething(int x, int base){
 // do something with x...
}

public static int doSomething(int x){
  return doSomething (x, 10);
}
```

העמסת פונקציה עם מספר ארגומנטים שונה

```
public static double average(double x){
 return x;
}

public static double average(double x1, double
 x2){
 return (x1 + x2) / 2;
}

public static double average(double x1, double x2,
 double x3){
 return (x1 + x2 + x3) / 3;
}
```

- יש שכפול לקוד
- לא תומך בממוצע של 4 ארגומנטים או יותר

העמסת פונקציה עם מספר ארגומנטים שונה- פתרון

```
public static double average(double [] args) {
 double sum = 0.0;
 for (double d : args) {
    sum += d;
 }
 return sum / args.length;
}
public static void main(String[] args) {
 double [] arr = {1.0, 2.0, 3.0};
 System.out.println("Averge is:" +
 average(arr));
```

- יש צורך ליצור מערך!!

העמסת פונקציה עם מספר ארגומנטים שונה- פתרון

- **פתרון של JAVA שימוש בשלוש נקודות.**
- תחביר מיוחד של שלוש נקודות (...) יוצר את המערך מאחורי הקלעים:

```
public static double average(double ... args) {
 double sum = 0.0;
 for (double d : args) {
    sum += d;
 }
 return sum / args.length;
}
```

- ניתן כעת להעביר לשרות מערך או ארגומנטים בודדים:

```
double [] arr = {5.0, 7.0, 8.0};
System.out.println("Averge is:" + average(arr));
System.out.println("Averge is:" + average(2.0, 2.0, 7.0));
```

הגדרת בנאי - constructor

- בכל מחלקה מוגדרות פונקציות אתחול (אחת לפחות) ששמן כשם המחלקה. פונקציות אלו נקראות **בנאים**
- כמו שצוין מקודם - אופרטור ה- **new** מקצה זיכרון לאובייקט החדש ומאתחל אותו ע"י קריאה לבנאי
- פעולה בונה יכולה לקבל פרמטרים, פרמטרים אלה משמשים לאתחול תכונות העצם.

הגדרת בנאי - constructor ושימוש בהעמסה

- בשימוש בעיקרון ההעמסה ניתן להגדיר יותר מבנאי אחד למחלקה (לשים לב לשימוש ב this כאשר השמות דומים).

```
public class Time {
        private int hour=0;
        private int minute=0;
        //empty constructor
        public Time()
        {
        }
//constructor initializing hour and minutes
        public Time(int hour,int minute)
        {
                this.hour = hour;
                this. minute = minute;
        }
}
```

הפונקציה toString

- מחזירה מחרוזת לצורך הדפסה לתיאור המחלקה. למשל המחלקה Date ניתן להוסיף:

```
public String toString()
{
        return "The date is " +
this.dd+"/"+this.mm+"/"+this.yy;
}
```

בהנחה ו- date1 מסוג Date ומכיל ערכים, ניתן להדפיס אותו בצורה הבאה:

- System.out.println(date1);
- הקומפיילר יפעיל את הפונקציה גם אם לא נכתוב קריאה ישירה אליה.

דוגמא:

- לדפנה יש 3 חתולים –mush, juke, dina.
- החתולים mush ו- juke הינם חתולי כביש והחתולה dina היא חתולה פרסית.
- Juke ו- dina אוהבים לאכול milky. mush נוהג ליילל.
- יש להגדיר ולממש את המחלקה Cat. המחלקה צריכה להכיל פעולה בונה המאתחלת את תכונות העצם.
- פעולה likeMilky המחזירה true אם החתול אוהב מילקי אחרת מחזירה false.
- פעולה isNoisy המחזירה את המחרוזת "Meow Meow Meow" עבור חתול שנוהג ליילל אחרת מחזירה את המחרוזת "Meow".
  פעולה toString המחזירה את המחרוזת
  "My name is <cat name> and I am <cat type>"

פתרון

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Class for defining the Object Cat
 * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
Public class Cat {
        private String name;
        private String type;
        private boolean milky;
        private boolean noisy;
        //constructor receiving cat attributes
        public Cat(String name, String type, boolean milky, boolean noisy)
        {
                this.name = name;
                this.type = type;
                this.milky = milky;
                this.noisy = noisy;
        }
        //method to see if cat like milky
        public boolean likeMilky()
        {
                // return value saved in milky attribute
                        return this.milky;
        }
```

פתרון - המשך

```
//method to see if cat is noisy
        public String isNoisy()
        {
                //check if noisy using the noisy attribute
                if(this.noisy == true)
                        return "Meow Meow Meow";
                else
                        return "Meow";
        }
        //method to return cat name and type
        public String toString()
        {
        return "My name is " + this.name + " and I am " + this.type;
        }
}
```

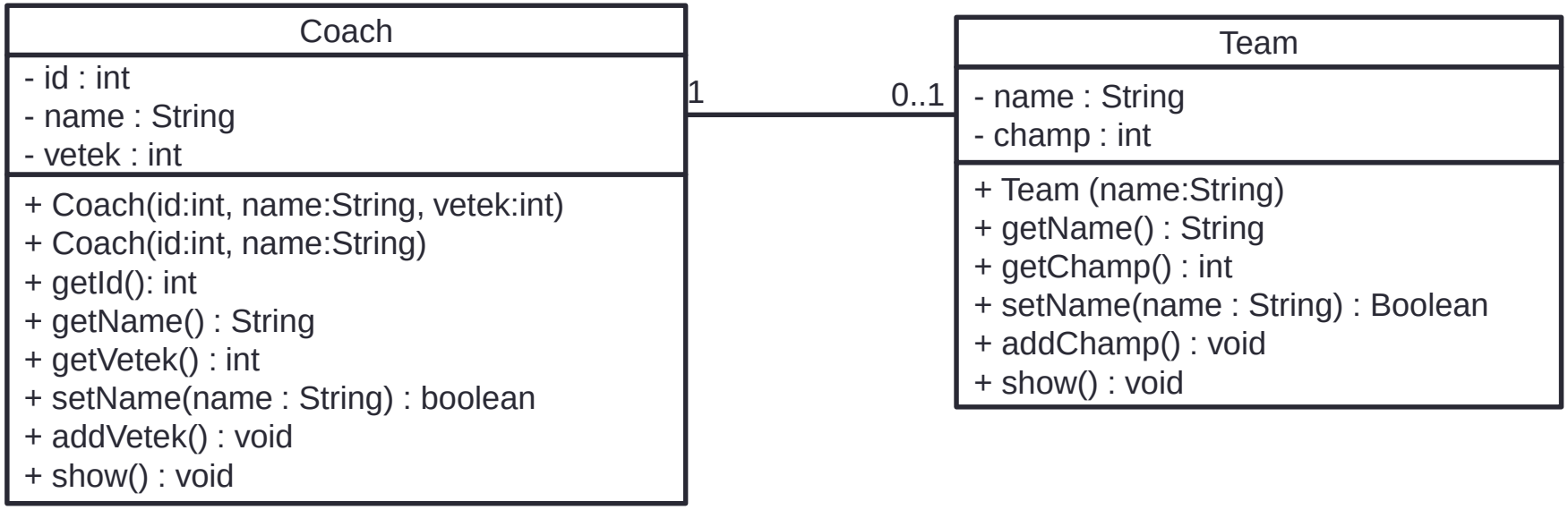
פתרון - המשך

המחלקה הראשית:

```
Import  java.util.Scanner;
public class Main {
        publicstaticvoid main(String[] args) {
                Cat[] daphneCats = new Cat[3];
                String catName, catType;
                boolean likeMilky, isNoisy;
                Scanner input = new Scanner(System.in);
                //insert cat info
                for (int i=0; i<3; i++)
                {
                        System.out.print("Enter cat name, type, like milky?, is noisy?");
                        catName = input.next();
                        catType = input.next();
                        likeMilky = input.nextBoolean();
                        isNoisy = input.nextBoolean();
                        daphneCats[i] = new Cat(catName, catType, likeMilky, isNoisy);
                }
                //show if second cat like milky
                System.out.print(daphneCats[1].toString() + " I like milk? ");
                System.out.print(daphneCats[1].likeMilky());
        }
}
```

מחלקות – קשר יחיד ליחיד

- בכל מחלקה קיים מצביע למחלקה השניי



מימוש המחלקה Team

```
public class Team {
        private String name;
        private int champ;
        public Team(String
name){

        setName(name);
                champ=0;
        }
        public String
getName(){
                return name;
        }
        public int getChamp(){
                return champ;
        }
```

```
        public Boolean setName(String
name){
                this.name = name;
                return true;
        }
        public void addChamp(){
                champ++;
        }
        public void show(){
        System.out.println("name: " + name
                + " champ: " + champ);
        }
}
```

מימוש המחלקה Coach

```
public class Coach {
        private int id;
        private String name;
        private int vetek;
        public Coach(int id, String name, int vetek)
                this.id=id;
                setName(name);
        this.vetek=vetek;
        }
        public Coach(int id, String name){
                this(id,name,0);
        }
```

```
        public int getId(){
                return id;
        }
        public String getName(){
                return name;
        }
        public void getVetek(){
                return vetek;
        }
```

המשך – מימוש המחלקה Coach

```
        public Boolean setName(String name){
                this.name = name;
                return true;
        }
        public void addVetek(){
                champ++;
        }
        public void show(){
        System.out.println("id: " + id + " name: " + name
                + " vetek: " + vetek);
        }
}
```

מימוש הקשר במחלקה Team

```
public class Team {
        private String name;
        private int champ;
        private Coach coach;
        …
        public Coach getCoach(){
                return coach;
        }
        public boolean setCoach(Coach c){
        //TODO
        }
}
```

- נוסיף למחלקה Team אובייקט מסוג Coach
- נוסיף למחלקה פעולת getCoach אשר תחזיר את ה- coach
- נוסיף למחלקה פעולה setCoach אשר תקבל מאמן לעדכון – מימוש בהמשך

מימוש הקשר במחלקה Coach

```
public class Coach {
        private int id;
        private String name;
        private int vetek;
        private Team team;
        …
        public Team getTeam(){
                return team;
        }
        public boolean setTeam(Team t){
        //TODO
        }
}
```

- נוסיף למחלקה Coach אובייקט מסוג Team
- נוסיף למחלקה פעולת getTeam אשר תחזיר את ה-team
- נוסיף למחלקה פעולה setTeam אשר תקבל קבוצה לעדכון – מימוש בהמשך

מימוש פעולת setTeam

```
public class Coach {
        …
        public boolean
setTeam(Team t){
        if(team!=t){
                team = t;
                if(team!=null)
                team.setCoach(this)
        ;
                return true;
        }
        return false;
        }
}
```

- הבדיקה הראשונה בודקת שהקבוצה החדשה איננה הקבוצה הנוכחית על מנת למנוע לולאה אינסופית, אם כן:
  - משייכים את הקבוצה
  - בודקים שהמאמן לא פוטר (כלומר, לא קבוצה ריקה) ואם כן:
    - מעדכנים את הקבוצה החדשה שזהו המאמן שלה

מימוש פעולת setCoach

```
public class Team {
        …
        public boolean setCoach(Coach c){
        if(this.coach != c){
                if(this.coach != null)
                 this.coach.setTeam(null);
                this.coach = c;
                this.coach.setTeam(this);
        return true;
        }
        return false;
        }
}
```

- הבדיקה הראשונה בודקת שהמאמן החדש איננו המאמן הנוכחי על מנת למנוע לולאה אינסופית, אם כן:
- בודקים אם היה קיים מאמן אחר ואם כן:
  - מיידעים אותו שהוא מפוטר
- משייכים את המאמן
- מעדכנים את המאמן החדש שזוהי הקבוצה שלו

Inheritance הורשה

- שפת Java מאפשרת לנו להגדיר מחלקות ראשיות (הורה) ומחלקות משניות (ילדים)שיורשות את התכונות של המחלקות הראשיות. את ההגדרה של מחלקה יורשת נציין ע"י המילה השמורה extends.
- לדוגמא ירושה בתוך מכללה:

```
publicclass Person {
        . . .
}
publicclass Student extends Person{
        . . .
}
publicclass Employee extends Person{
        . . .
}
publicclass Lecturer extends Employee{
        . . .
}
```

הסבר הדוגמא

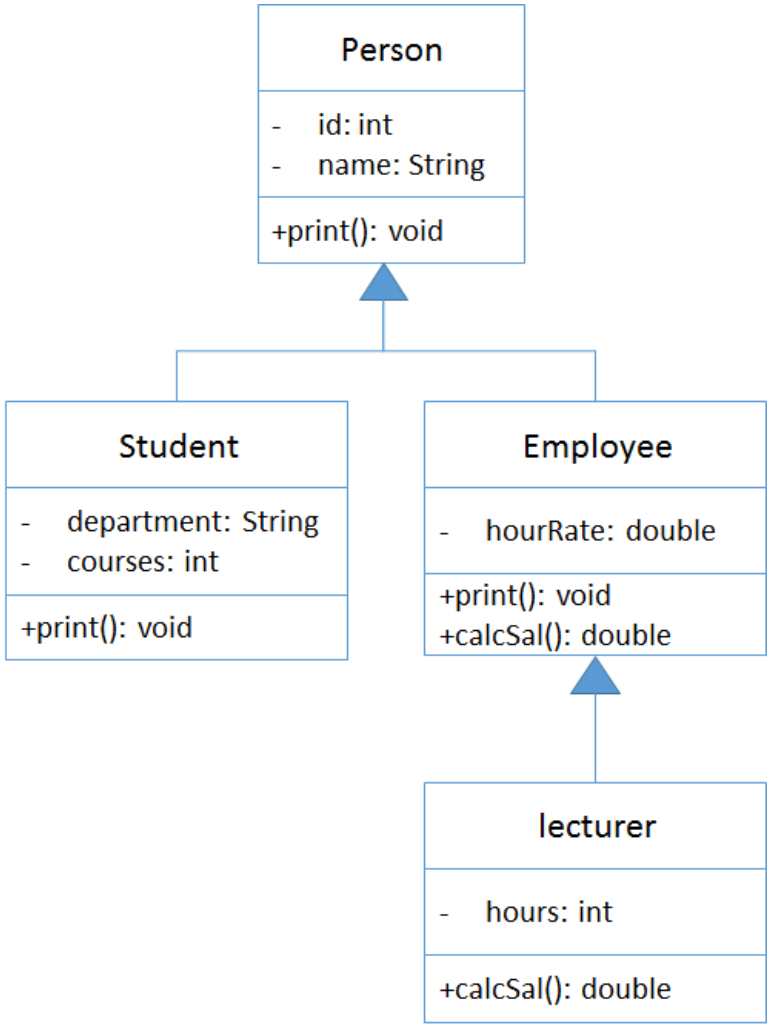
- בדוגמא יש 4 מחלקות:

א. המחלקה Person שהינה מחלקת הבסיס המייצגת נתונים של ישות אנושית השייכת למכללה (תז, שם וכו').

ב. 2 מחלקות בנים שירשו את המחלקה Person– המחלקה Student שתייצג תכונות נוספות הקשורות בסטודנטים (שם מגמה, חובות כספיים, קורסים נלמדים וכו') והמחלקה Employee שתייצג תכונות נוספות הקשורות לעובדי המכללה (סוג המשרה, שעות עבודה, משכורת וכו').

ג. למחלקת Employee בעצמה יש מחלקת בת שיורשת אותה – המחלקה Lecturer שמייצגת תכונות נוספות הקשורות במרצים (תארים, קורסים מועברים וכו').

מבנה גרפי לדוגמא:



הורשה - יתרונות

- כל עצם מסוג בן שניצור יקבל גם את תכונות העצם מסוג אב שלו. הירושה:
- מקלה על ההבנה – דימוי העולם האמתי בעת כתיבת תכנית.
- מאפשרת כתיבה קצרה יותר – אין צורך לכתוב תכונות זהות לכל מחלקה ומחלקה מספיק להגדיר מחלקת בסיס משותפת.
- מאפשרת להרחיב את הקוד הקיים כבר במחלקות הספרייה.
- יצירת ממשק אחיד לעצמים ממחלקות נגזרות – פולימורפיזם (בהמשך).

המילה השמורה - super

- באמצעות המילה השמורה super ניתן לגשת ממתודה במחלקה יורשת למתודה במחלקת הבסיס. למשל:

```
public class Person {
        private int id;
        private String name;
        public Person(int id, String name)
        {
                this.id = id;
                this.name = name;
        }
        public void print()
        {
                System.out.print("ID: " + this.id);
                System.out.print(" Name: " + this.name);
        }
}
```

המילה השמורה - super – המשך דוגמא

```
public  class Student extends Person{
        private String department;
        private int courses;

        public Student(int id, String name, String department, int courses)
        {
                super(id, name);
                this.department = department;
                this.courses = courses;
        }
        public void print()
        {
                super.print();
                System.out.print(" Department: " + this.department);
                System.out.print(" Courses: " + this.courses);
        }
}
```

המילה השמורה - super – המשך דוגמא

```
public class Main {

        public static void main(String[] args) {

                Student s1;
                s1=new Student(123,"Shadi","Software",22);
                s1.print();
        }
}
```

הסבר דוגמא

- המחלקה Person כוללת שתי תכונות (תז ושם) ושתי מתודות (בונה והדפסה).
- המחלקה יורשת ממחלקת Person ומוסיפה לה שתי תכונות (שם מגמה, מספר קורסים) כל עצם שניצור מסוג Student מורכב מארבעה תכונות, שתי תכונות ממחלקת הבסיס ושתי תכונות של המחלקה עצמה. למחלקה שתי מתודות, פעולה בונה, שבה יש שימוש במילה השמורה super כדי לפנות לפעולה הבונה של מחלקת הבסיס על מנת להזין את שתי התכונות הראשונות (תז, שם) לאובייקט, ופעולת הדפסה שבה יש באופן דומה שימוש במילה super כדי לפנות למתודת ההדפסה של מחלקת הבסיס.

כללי תורשה

- כל תכונה שקיימת במחלקת הבסיס מועברת בירושה למחלקה היורשת. המחלקה הנגזרת מגדירה תכונות נוספות לעצמה, בלי קשר למחלקת הבסיס.
- מחלקה יורשת יכולה להגדיר פונקציות חדשות, גם בעלות שם זהה לפונקציות שניתנו ע"י מחלקת הבסיס, וכך לדרוס (override) את אותן פונקציות.
- תמיד בקריאה לפונקציה תיבחר הגירסה העדכנית ביותר שלה.
- אין אפשרות לרשת מכמה מחלקות שונות.

מניעת תורשה

- ניתן להגדיר מחלקה כאחרונה בסדר הירושה ע"י שימוש במילה השמורה final לדוגמא נגדיר את המחלקה lecturer כאחרונה:

```
public final class Lecturer extends Employee{
        . . .
}
```

- כלומר, לא ניתן להגדיר מחלקה שתירש את מחלקת lecturer.
- בדומה ניתן להגדיר מתודה כלא ניתנת לדריסה (override) ע"י שימוש באותה מילה השמורה. לדוגמא:

```
public class Employee extends Person{
        . . .
        final void print()
        {
                . . .
        }
}
```

- ע"י הגדרת המתודה print במחלקת employee כ- final לא נוכל לדרוס אותה ולהגדיר מתודת הדפסה למחלקת lecturer

תרגיל

- תכנן מערכת שמכילה לפחות 4 מחלקות. אחת ראשית והשאר יורשות ממנה, וחלק יורשות מעצמן. (לעשות לפחות שתי רמות של תורשה).
- ממש את פונקציות ה set/get.
- שתי מחלקות צריכות להיות בקשר יחיד ליחיד.
- קלוט נתונים ב main ראשי של המחלקות היורשות.
- בחר לך ערך מספרים שהוא מהווה חיתוך לנתונים מסוימים. לדוגמא: אם המחלקה היא סטודנטים ויש לסטודנט ממוצע ציונים, אז הדפס את פרטי כל הסטודנטים בעלי ממוצע 85.