הרצאה מס' 3

הורשה Inheritance

נתחיל עם דוגמה,

נבנה מחלקה חדשה בשם Person:

```
public class Person {

	protected String name;
	protected long id;
	protected char gender;

	public Person(String name, long id, char gender) {
		this.name = name;
		this.id = id;
		this.gender = gender;
	}

	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public char getGender() {
		return gender;
	}
	public boolean setGender(char gender) {
		if(gender != 'M' && gender !='F')
			return false;

		this.gender = gender;
		return true;
	}

	public String toString() {
		return "Person [name=" + name + ", id=" + id + ", gender=" +		gender+
	"]";
	}
	public void draw ()
	{
		if(this.gender == 'F')
		{
			System.out.println(" :)");
		}
```

```
                else
                        System.out.println(" :(");

        }
}
public static void main(String[] args) {

        Person p = new Person("Yossi",111,'M');
        Person theWife = new Person("Miri",222,'F');
        System.out.println("This Is Yossi");
        p.draw();
        System.out.println("And his lovely wife");
        theWife.draw();

        System.out.println("Testing copy");
        Person copy = new Person(p.getName(),p.getId(),p.getGender());
        copy.setName("Yossef");
        System.out.println (p);  // println.(p.toString())
                System.out.println(theWife);//println.(theWife.toString())

}
```

<u>הערות עיקריות:</u>

- **הגנה על משתנים פנימיים:**

בדרך כלל מומלץ להגדיר משתנים פנימיים כ private או protected ולגשת אליהם באמצעות מתודות get,set. היתרונות:

- מתאפשרת בדיקת תקינות ב set (למשל: לא מאפשר מספר ת.ז. שלילי)
- יתכן ששינוי בשדה אחד דורש עדכון בשדות אחרים (רעיון שלא מומש כאן: אולי אם "יוסף" משנה את שמו ל "יוספה" , כדאי לברר אם נדרש עדכון נוסף?)
- שינוי בייצוג הפנימי של מחלקה לא ישפיע על קוד שמשתמש במחלקה.

- **הגבלות גישה למשתני/ מתודות מחלקה:**
  - Public – ניתן לגשת למשתנה / מתודה מכל מקום.
  - Private – ניתן לגשת רק מתוך מתודות המחלקה.
  - Protected – ניתן לגשת רק מתוך מתודות המחלקה או מחלקות הנורשות ממנה ללא תווית- package - ניתן לגשת רק מתוך מחלקות הנמצאות באותו package .

- **Constructor (מתודה בונה):**
  - מתודה זו נקראת כשנוצר אובייקט חדש, על ידי המילה new למשל:

```
Person p = new Person("David Cohen",12345,'M');
```

  - זו קריאה ל-

```
public Person(String name,long id,char gender)
```

- שם ה- constructor זהה לשם המחלקה.
- Constructor היא המתודה היחידה שאין לה ערך מוחזר (אפילו לא void).

- **המילה this:**
  - היא הפניה לאובייקט עצמו.
  - בהמשך נראה שהיא שימושית, כשהאובייקט נדרש "לשלוח את עצמו" (כתובתו) למחלקות אחרות.
  - בדוגמא, השתמשנו ב thisכאשר שם הפרמטר המגיע למתודה, היה זהה לשם השדה הפנימי של האובייקט (אי- הבנה שיכולנו למנוע מראש ע"י בחירת שם אחר לפרמטר המגיע למתודה).

נבנה מחלקה חדשה Student המכילה את כל הפרטים שמכילה המחלקה Person , ובנוסף לכך את השדות average-ıdepartment . נממש אותה תוך בהורשה.

```
public class Student extends Person {

	protected double average;
	protected String department;
	// constructor
	public Student(String name, long id, char gender, double  average ,String department) {
		super(name, id, gender);
		this.average = average;
		this.department = department;
	}
	//getters and setters
	public double getAverage() {
		return average;
	}
	public boolean setAverage(double average) {
		if (average < 0 ) return false;
		this.average = average;
		return true;
	}
	public String getDepartment() {
		return department;
	}
	public void setDepartment(String department) {
		this.department = department;
	}

	// to string
	public String toString() {
		return "Student [average=" + average + ", department=" + department + ", gender="
+ gender + ", id=" + id + ", name=" + name + ", getAverage()=" + getAverage()
+ ", getDepartment()=" + getDepartment() + ", getGender()="
+ getGender()
+ ", getId()=" + getId() + ", getName()=" + getName()
+ ", toString()=" + super.toString() + ", getClass()="
+ getClass() + ", hashCode()=" + hashCode() + "]";
	}

	// draw method
	public void draw()
	{
		if(this.gender =='M') System.out.println(" :))");
		if(this.gender == 'F') System.out.println(":((");
	}
 }

public class StudentTest {
	public static void main(String[] args) {
		Student s = new Student("Noa,Zelger" ,34222444,'F', 85.3, "Math");
		s.setAverage(91.5);
		s.setName("Noa Zelger- Avinoam");
		System.out.println(s);
		s.draw();
		float av= (float)s.getAverage();

		if(av >= 90) System.out.println("Well Done");
	}
}
```

נוסיף מחלקה נוספת, ליצן שגם היא יורשת מאדם.

```
public class Clown extends Person {

	protected String joke; //his-her best joke;
	protected String circus;

	// constructor
	public Clown(String name, long id, char gender, String joke, String
circus) {
		super(name, id, gender);
		this.joke = new String(joke);
		this.circus = new String(circus);
	}
	//getters and setters
	public String getJoke() {
		return joke;
	}
	public void setJoke(String joke) {
		this.joke = joke;
	}
	public String getCircus() {
		return circus;
	}
	public void setCircus(String circus) {
		this.circus = circus;
	}

    //to String
	public String toString() {
		return "Clown [circus=" + circus + ", joke=" + joke + ", gender="
				+ gender + ", id=" + id + ", name=" + name + ",
getCircus()="+ getCircus() + ", getJoke()=" + getJoke() + ", getGender()="
+ getGender() + ", getId()=" + getId() + ", getName()="
+ getName() + ", toString()=" + super.toString()
+ ", getClass()=" + getClass() + ", hashCode()=" + hashCode()
				+ "]";
	}
	public void draw()
	{
		System.out.println("< :)");
	}


}
```

```
public class ClownTest {

	static void main(String[] args) {
		String joke = "Why did the chicken cross the road? \n" + "to get the
other side";

		Clown c = new Clown("Bozo",333333,'M',joke,"the great circus");
		c.setId(7777777);
		System.out.println(c);
		c.draw();
	}
}
```

אבחנות עיקריות :

- הורשה משמשת כאשר מחלקה אחת היא מקרה פרטי של מחלקה אחרת. למשל: ליצן הוא **מקרה פרטי (סוג)** של אדם, לכן סטודנט יורש אדם .
- Java מאפשרת **לרשת ממחלקה אחת לכל היותר**. למשל: אם הגדרתם מחלקה Student ומחלקה Employee, אזי לא נוכל להשתמש בextends להגדרת "סטודנט שהוא גם עובד".

~~Class WorkingStudent extends Student, Employee {…~~

- ניתן לייצור כמה דרגות הורשה , למשל:

```
Class Animal {}
Class Mammal extends Animal {}
Class Person extends Mammal {}
```

- כשאובייקט (בן) נורש מאובייקט (ב), הוא יורש את המתודות של האב:

לדוגמא:

```
Clown c = new Clown();
System.out.println("c.getName());
```

ליצן יורש (משתמש ב-) הפונקציה ()getName של אדם.

- אם צריך, ניתן **לדרוס (להגדיר מחדש)** מתודה של אב.

לדוגמא: מאחר שהחלטנו שליצנים נראים אחרת מאדם רגיל, דרסנו את ()draw (אלמלא הדריסה, היה הליצן יורש את draw ומצטייר כאדם רגיל).

- **המילה Supper:**

מאפשרת לבן לגשת למתודות של האב. בפרט: קריאה ל constructor של האב (אם רוצים לבצע קריאה כזו, היא חייבת להופיע **בשורה הראשונה** ב constructor של הבן) :

```
public Clown (String name,lng id,char gender,String joke, String circus)
{
        super(name,id,gender);
        this.joke = joke;
        this.circus = circus;
}
```

- הערה: אם לא קוראים במפורש ל- super, תתבצע קריאה אוטומטית לconstructor הריק של האב (כלומר, הקריאה: ()super).
- super מאפשר גישה לפונקציות של האב, גם אם הן נדרסו ע"י הבן. למשל:

ב Clown הגדרנו ()draw לציור עם כובע ליצן.

אם הליצן שלנו יוצא לחופשה ומבקש להראות כאדם רגיל, נכתוב:

```
class Clown extend Person
{
        ….
        public void drawAfterWork()
        {
                super.draw();
        }
}
```