# פולימורפיזם

ב- Java ניתן להתייחס לעצמים שונים בתור דברים דומים. למשל, ישנם סוגים שונים של מחשבים אישי, נייד, כף יד, שרת וכו'. ניתן להגדיר כל אחד מהם בנפרד אך ניתן להתייחס אליהם בתור מחשב, כלומר, למרות שהם שונים יש ביניהם מכנה משותף.

בדוגמת המכללה הגדרנו סוג בסיסי person ושלושה סוגים נגזרים student, employee, lecturer. ניתן ליצור מערך שיכיל את כל הישויות שקיימות במכללה ע"י שימוש בסוג הבסיס וכך יבוצע שימוש בעיקרון הפולימורפיזם שלפיו:

- מצביע על עצם בסיס יכול להצביע בהתאם גם על עצם של מחלקה שנגזרת ממחלקת הבסיס.
- כל מתודה ב- Java מוגדרת כמתודה וירטואלית, כלומר, המתודה תקרא לעצם לפי מה שהוא באמת ולא תלך למתודה של מחלקת הבסיס.

דוגמא והסבר: מקודם ביצענו את הפעולות הבאות:

- במחלקה Person הגדרנו draw המצייר אדם רגיל.
- במחלקה Clown דרסנו את draw כך שיצייר ליצנ/ית עם כובע.

אם תוסיפו main עם השורות הבאות תקבלו תוצאות צפויות:

```
Person p = new Person("yossi",1111,'M'); // בלי כובע

p.draw();

Clown c  = new Clown(("yossi",1111,'M',"ha","The Great Circus");

c.draw();
```

אבל מה יקרה אם נוסיף:

```
Person x = new Clown(("yossi",1111,'M',"ha","The Great Circus");

x.draw();
```

מה זה?!

1. האם זה בכלל חוקי?
2. גם אם כן, למה שמישהו יתעקש לכתוב כך?
3. האם X חושב שהוא אדם רגיל, ו ליצן? האם יצויר עם כובע או בלעדיו?

תשובות:

1. זה חוקי! ההיגיון: בהצהרה "Person x;" הודענו "X צריך להיות אדם כלשהו". אז בפרט, מותר לו להיות ליצן.

ההפך אינו חוקי- אסור לכתוב ~~Clown c = new Person();~~ כי ההצהרה "Clown c", פירושה שדרוש ליצן, שיכול לספר בדיחות- אם פשוט לא יספיק.

2. גישה זו תועיל ב:

- מצבים בהם אנחנו מצפים לקבל "אדם כלשהו", ולא חשוב אם יגיע אדם פשוט, או ליצן, או סטודנט וכו', למשל:

```
class  Dog{
    protected Person owner; //אדם כלשהו- יתכן ליצן או סטודנט
}
```

או תור בבנק, מערך של בני אדם, כשכל אחד יכול להיות ליצן או סטודנט:

```
Person[] people;
```

3. - X זוכר שהוא ליצן!

- Draw היא מתודה שתקרא בהתאם לאובייקט שעליו הפעלנו אותה ולא בהתאם לרפרנס. ולכן x.draw() יצייר ליצן (עם כובע ואף מצחיק).
- ובאופן כללי, מידע אינו אובד בהצבה הזו (גם שדה "קרקס" לא נמחק, ולא נמחקה היכולת לספר בדיחה, אבל נדרש casting כדי להגיע אליהם).

הדבר מתאפשר כי עובדים עם מצביעים (לא "דוחסים 5 שדות לתוך שלושה!")

**דוגמא (פולימורפיזם): תור של אנשים**

```
//Queue Test
public class QueueTest {

	public static void main(String[] args) {
		Person[] queue = new Person[4];

		queue[0] = new Person("Miri",2222,'F');
		queue[1] = new Student("Zehava",1111,'F',95.7,"Collman");
		queue[2] = new Clown("Bozo",33333,'M',"bla ha ha","The curcis");
		queue[3] = new Person("Dani",4444,'M');

		for(int i=0; i<queue.length;i++)
		{
			queue[i].draw();
		}
	}
}
```

**דוגמה נוספת:**

מה יודפס בכל שורה בקוד הבא?

```
Person[] queue = new Person[3];

queue[0] = new Person("Miri",2222,'F');
queue[1] = new Student("Zehava",1111,'F',95.7,"Collman");
queue[2] = new Clown("Bozo",33333,'M',"bla ha ha","The curcis");

System.out.println("queue[0].setName('Miriam'):");
queue[0].setName("Miriam");

System.out.println("queue[1].draw():");
queue[1].draw();

System.out.println("queue[2].tellJoke():");
queue[2].tellJoke();
```

האם קיימת בעיה בקוד?

כן! אדם queue[2] אינו יודע לספר בדיחות! ההתייחסות אליו בתוך המערך היא התייחסות לאדם. ובמחלקה Person לא מוגדרת מתודה tellJoke().

כיצד נפתור את הבעיה?

```
System.out.println("queue[2].tellJoke():");
((Clown)queue[2]).tellJoke();
```

**<u>הסבר מדויק יותר:</u>**

אם הגדרנו Person p = new Clown() ניתן לחשוב על שלושה סוגי פונקציות:

**א. כאלה המוגדרות רק באדם, למשל:**

```
String s = p.getName();
```

<u>זה עובד,</u> מאחר שליצן יורש פונקציות כאלה מאדם.

**ב. כאלה המוגדרות באדם, ונדרסות (מוגדרת מחדש) בליצן, למשל:**

```
p.draw();
```

זה מתקמפל, וצייר ליצן (הקומפיילר מאפשר זאת, מפני ש draw מוגדרת ב"אדם" ולכן מובטח שתהייה כזו. ואילו בזמן ריצה התוכנית מוודאת את הסוג המדויק של P וקוראת לפונקציה המתאימה ביותר.)

**ג. כאלה המוגדרות רק עבור ליצן:**

```
p.tellJoke();
```

זה לא יתקמפל, מפני ש p הוצהר Person רגיל, שאין לו אפשרות לספר בדיחה (חשבו במיוחד על מקרה בו Person p מוצהר בתחילת התוכנית, ורק בזמן ריצה מחליטים אם לשים בו ליצן או סטודנט..).

פתרון: "להזכיר " שהוא ליצן, על ידי casting של המצביע:

```
((Clown)queue[2]).tellJoke();
```

**Instanceof**

בדוק האם אובייקט שייך למחלקה (או למחלקה הנורשת ממנה).

דוגמא: אילו הדפסות יבוצעו (אילו תנאי if נכונים)

```
Person p = new Person();
Student s = new Student();
String str  = new String("hello");
if(p instanceof Person)
        System.out.println("p is person");
if(p instanceof Student)
        System.out.println("p is Student");
if(s instanceof Person)
        System.out.println("s is person");
if(str instanceof Person)
        System.out.println("str is person");
if(s instanceof Student)
        System.out.println("s is Student");
```

# מודל ההכלה

ישנם כמה סוגי מחלקות אותן ניתן להגדיר בתוך מחלקה אחרת.

<u>שים♡:</u>

- במחלקה פנימית אי אפשר להגדיר מתודות או תכונות סטטיות
- מחלקה חברה תקבל שם שונה משם המחלקה המכילה
- בזמן הידור (compile) ייבנה קובץ class מיוחד למחלקה המוכלת, שמו יהיה המחלקה המכילה $ המחלקה המוכלת לדוגמא: Father$Son.class

---

מחלקה סטטית

האפשרות היחידה להגדיר מחלקה פנימית שתהיה רגילה לחלוטין בשימוש שלה היא ע"י הגדרת מחלקה סטטית בתוך המחלקה המכילה. הקשר של המחלקה המוכלת למחלקה המכילה יהיה בקריאה למחלקה ולמתודות שלה אשר במרחב השמות של המחלקה המכילה.

לדוגמא:

```
public class Father{
      . . .
      private int x;
      . . .
      static class Son
      {
            private int x;
      }
}
```

בכל אחת xנוצרו שתי מחלקות כאשר מחלקת האב מכילה את מחלקת הבן אך אין קשר בין המחלקות והתכונה מהמחלקות הינה תכונה בפני עצמה. שימוש בעצמים יבוצע בצורה הבאה:

```
public class A {
      public static void main(String[] args) {
            Father f = new Father();
            Father.Son s = new Father.Son();
      }
}
```

כלומר, יצירת עצם מסוג מחלקת Son מתבצעת תוך שימוש במרחב השמות של מחלקת Father ש- Son מוכלת בתוכה.

---

מחלקה חברה

מחלקה שמוגדרת בתוך מחלקה אחרת בלי להיות סטטית. אי אפשר להגדיר עצם מהמחלקה אלא רק כחלק מעצם שהוגדר למחלקה המכילה. שימוש באיברי המחלקה יהיה דרך מרחב השמות של המחלקה המכילה.

לדוגמא:

```
public class Father{
	int a =1;
	class Son{
		int a = 2;
		class Grandson{
			int a = 3;
			void out()
			{
				System.out.println ("a = " + a);
				System.out.println ("a = " + Son.this.a);
				System.out.println ("a = " + Father.this.a);
			}
		}
	}
}
```

הקריאה מהמחלקה הראשית:

```
public class FriendClass{
	public static void main(String[] args)
	{
		Father f = new Father();
		Father.Son s = f.new Son();
		Father.Son.Grandson g = s.new Grandson();
		g.out();
	}
}
```

הפלט של התוכנית יהיה:

```
a =3
a =2
a =1
```

מאחר ואי אפשר ליצור עצם של מחלקה פנימית בצורה חופשית אלא רק כחלק מעצם של המחלקה שמכילה אותה, יצירת העצם של מחלקה פנימית תיעשה ע"י קריאה לעצם שכבר יצרנו מסוג המחלקה המכילה ודרכו ניצור את העצם החדש של המחלקה המוכלת.

בקריאה לתכונות במתודה out מאחר והן בעלות שם זהה הקריאה נעשתה ע"י שימוש בשם המחלקה שהתכונה שייכת אליה ושימוש במצביע העצמי this.

**אמא יש רק אחת**

ב java מוגדרת מחלקה **Object** שהיא אב קדמון של כל מחלקה אחרת.

עובדה זו, בשלוב עם פולימורפיזם, מאפשרת הגדרת **מבני נתונים כללים**.

```
public class MyObjects {
private Object[] arr;

	public MyObjects(int size){
		arr = new Object[size];
	}
	public Object getElementAt(int index){
		return arr[index];
	}

	public void setElementAt(int index, Object obj){
		arr[index] = obj;
	}
	public int getLength(){
		return arr.length;
	}

	public void print(){
		for(int i=0;i<arr.length;i++) // assuming toString is properly
overridden
			System.out.println(arr[i]);
	}
	public Object find(Object obj){
		for(int i=0;i<arr.length;i++) // assuming equals is properly overridden
			if(arr[i].equals(obj)) return  arr[i];
		return  null;
	}

}
public class Program {
	public static void main(String[] args) {
			String s = "hello";
			Person p2 = new Person("Yossi",11111,'M');
			MyObjects  ma  = new MyObjects(3);
			ma.setElementAt(0,s);
			ma.setElementAt(2, p2);
			Object o = ma.find("hello");
			if((o != null) && (o instanceof String)){
				String s2 = (String)ma.getElementAt(0);//casting required
				System.out.println(s2);
				String s3  = (String)o; // casting required
				System.out.println(s3);
			}
	}
}
```

**נושאים**:

- Abstract
- interfaces (ממשקים).

## Abstract

מחלקה אבסטרקטית הינה מחלקה המשמשת כתבנית למחלקות היורשות אותה.
אין סיבה ואף אסור ליצור מופע (אובייקט) מסוג מחלקה זו.
שיטה/ מתודה שמוגדרת כאבסטרקטית חייבת להיות ממומשת ע"י המחלקות היורשות. אחרת מחלקה שלא תממש תהפוך בעצמה לאבסטרקטית.

דוגמא:

```
abstract class Animal {

        protected int legs; // number of legs

        public Animal(int legs)
        {
                this.legs= legs;
        }
        public int getLegs() {
                return legs;
        }

        public void setLegs(int legs) {
                if(legs >=0)
                        this.legs= legs;
        }

        abstract void draw();  //אבסטרקטי – מימוש  ללא
        abstract void speak(); //אבסטרקטי- מימוש  ללא
}

//-----------------------------------------------------------------
public class Dog extends Animal
{
        public Dog(){ super(4); }
        void speak(){ System.out.println("Wooooof");}
        void draw() { System.out.println("1-1^");}
}
//-----------------------------------------------------------------
public class Cat extends Animal
{
        public Cat(){ super(4); }
        void speak(){ System.out.println("Meawwwwwww");}
        void draw() { System.out.println("1-1*^");
}
```

**הסבר**:

אין משמעות להגדרת מופע של Animal בפני עצמה. אפשר לבקש ליצור כלב (או חתול), אבל אין משמעות לבקשה ליצור יצור בעל חיים כלשהו.

המחלקה Animal הוגדרה כאבסטרקטית, כלומר אסור ליצור מופעים שלה. אך כן ניתן ליצור מופעים של כלב או חתול.

בנוסף: החלטנו לדרוש שלכל חיה תהיה מתודת ציור ומתודת השמעת קול. הן יוגדרו בנפרד לכל חיה, אבל אין משמעות למימושן ב Animal הבסיסית.

לכן מתודות אלה הוכרזו כ abstract.

פירוש הדבר: אין להן שימוש ב Animal . אבל נדרוש לממשן בכל מחלקה שיורשת מ Animal

**הערה חשובה:**

מחלקה יורשת שלא תממש אותן, תהפוך בעצמה אבסטרקטית וכך שלא יתאפשר ליצור מופע שלה.

## ממשק Interface

Interface הוא מחלקה אבסטרקטית, שבעיקרה לא כוללת שום דבר מלבד רשימת שמות המתודות. מחלקה ש "תממש" את "הממשק" תתחייב לממש את כל המתודות ברשימה.

דוגמא:

```
public interface Noisy {
	void softSound();
	void loudSound();

}
//-----------------------------------------------------------------------------------------------------------------------------------
public class Cat implements Noisy
{
	protected String name;
	public Cat(String name)                 { this.name = name; }
	public String getName()           { return name;  }
	public void setName(String name) { this.name = name;}

	public void softSound(){ System.out.println("meaw...");}
	public void loudSound() { System.out.println(this.name);}
}
//-----------------------------------------------------------------------------------------------------------------------------------
public class Motorcycle implements Noisy
{
	protected String  licence;
	public Motorcycle (String licence) { this.licence = licence; }
	public String getlicence()  { return licence;  }
	public void setlicence(String licence) { this.licence = licence;}

	public void softSound(){ System.out.println("trrrrrrrr");}
	public void loudSound() { System.out.println("TRRRRRRRRRRRRR!!!!");}
}

public class Program {
	public static void main(String[] args) {
		Noisy[] noisyThings = new Noisy[3];
		noisyThings[0] = new Cat("Jerry");
		noisyThings[1] = new Motorcycle("55-616-04");
		noisyThings[2] = new Cat("Fluffy");
		for(int i=0;i<noisyThings.length;i++)
			noisyThings[i].loudSound();

	}
}
```

```
Jerry
TRRRRRRRRRRRRR!!!!
Fluffy
```

הערות:

1. ב Main יצרנו מערך של "דברים רועשים". כדי ליצור מהומה, עברנו על איברי המערך ובקשנו מכל אחד להשמיע ()loudSound. לצורך העניין לא היה אכפת לנו אם מדובר בחתולים, אופנועים או תערובת של שניהם.

   חשוב לנו רק שתהייה מוגדרת מתודה ()loudSound וזה מובטח לכל מחלקה המממשת את Noisy.

2. ב Java אין תורשה מרובה. מחלקה יכולה לרשת (extends) ממחלקה אחת בלבד. אבל היא יכולה לממש (implements) מספר לא מוגבל של ממשקים. למשל:

```
public interface Drawable
{
        public void draw();
}
public interface Noisy {
        void softSound();
        void loudSound();
}

//-----------------------------------------------------
public class Cat implements Noisy, Drawable
{

        public void softSound(){...}
        public void loudSound(){...}
        public void draw(){....}
        public String toString() {....}
}
```

3. ממשקים משמשים גם לכתיבת מתודות ומבני נתונים כללים, כאשר לא ידוע בדיוק אילו אובייקטים יוחזקו במבנה הנתונים, אבל נדרוש שיהיו להם פונקציות מסוימות.

**בואי נכיר איך מקבלים מספר רנדומלי ☺ ...**

```
public class Program {
        public static void main(String[] args) {
                double f = Math.random();
                System.out.println(f);
                int num;
                f = f * 10;
                num = (int) f;
                num = num;
                System.out.println(num);
        }
}
```

**תרגיל כיתה**

האוניברסיטה החליטה למחשב את מערכת המידע של העובדים. ישנן 3 קבוצות של עובדים: עובדים כללים, פקידות/מזכירות ומרצים.

מחלקה interface תכיל שתי פונקציה:

- חישוב תוספות, כך שאם העובד עם וותק מעל 5 שנים מחזירה 10% מהמשכורת. ואם פחות מחזירה 7% מהמשכורת.

מחלקת עובד כללי, שמממשת את ה interface הנתונים שישמרו לגבי עובד כללי הם:

- שם
- מספר תעודת זהות
- משכורת בסיסית
- וותק

**לשים לב** שהפונקציות יעברו לבן היורש, כולל אלו שמומשו דרך ה interface אצל מחלקת האב.

לגבי כל פקידה/ מזכירה יישמרו בנוסף:

- תפקיד
- שכר לשעות עבודה נוספות
- מספר שעות נוספות בחודש האחרון

לגבי כל מרצה יישמרו בנוסף

- תואר (מר=1, ד"ר=2, פרופ= 3)
- שם החוג (המחלקה) אליו הוא שייך
- רשימת הקורסים שהוא מלמד, מערך של מחרוזות.
- סכום הבונוס שהוא מקבל החודש

פעולות האפשריות על כל סוגי העובדים:

- הדפסת הפרטים

דרישות

- ממש את המחלקות הנ"ל, השתמש בעקרונות ה OOP שנלמדו.
- הגדר נכון את סוגי המתודות והמשתנים. לגבי כ"א מהמחלקות ממש בנוסף למתודות הנ"ל את ה constructor, מתודות גישה (get,set) ומתודה toString.
- צור main שבו יוצרו 5 עובדים מכל סוג והכניסו אותם למערך (בגודל 15).
- הגרילו 3 מרצים מתוך המערך והדפיסו אותם. בהגרלה יש להשתמש באובייקט Random.

בהצלחה!!!! ☺

שאדי עסאקלה