Inheritance
&
Polymorphism

SYNTAX
שפה מייצרת חשיבה

ירושה - מהי ומדוע?

- תכנות מונחה העצמים נותן יכולת להוריש ממחלקה אחת למחלקה אחרת.
- מטרה - למנוע חזרה על קוד ולייצג יחסים הגיונים בין המחלקות, יחסים התואמים לעולם האמיתי.
- בהורשה אנו מגדירים יחס של המשכיות - האם זה ?
- כלומר האם ההונדה שלי היא מכונית, האם הכלב הוא בעל חיים, האם ההמבורגר הוא מאכל. אם התשובה כן. ניתן לייצר ירושה בין המחלקות.
- ההורשה מאפשרת להעביר משתנים ופונקציות, מאב לבן ולנכד.

SYNTAX
שפה מייצרת חשיבה

כללי הורשה

- כל התכונות והפונקציות במחלקת הבסיס) שאינן (private מועברות למחלקה היורשת.
- המחלקה היורשת/נגזרת יכולה להגדיר תכונות ופונקציות נוספות.
- מחלקה יורשת יכולה לדרוס (override) את מימוש מחלקת הבסיס.
- תמיד בקריאה לפונקציה תיבחר הגרסה העדכנית ביותר שלה.
- בג'אווה ניתן לרשת ממחלקה אחת בלבד.
- כל המחלקות ב Java יורשות אוטומטית מ.Object -

SYNTAX
שפה מייצרת חשיבה

אופן הירושה בJava

- ירושה היא מנגנון המאפשר להגדיר את המשותף שבין מספר המחלקות במחלקה אחת.
- המחלקה בה מוגדר חלק המשותף נקראת **מחלקת בסיס (Basic class).**
- מחלקה יורשת ממחלקת בסיס נקראת **מחלקה נגזרת (Derived class).**

```
public class A
{  ...}
```

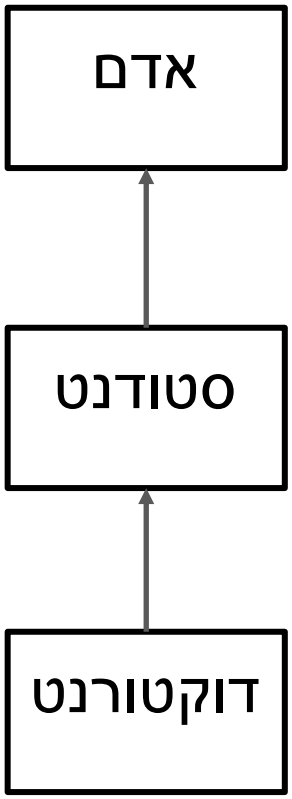
כדי לציין שמחלקה B יורשת מ A נכתוב

```
public class B extends A
{ ... }
```

- ירושה יוצרת יחס : "is a"
- המילה שמורה **extends** מציינת ירושה.

דוגמות לירושה – היררכית אוניברסיטה

- האם הסטודנט הוא גם אדם? האם הדוקטורנט הוא גם סטודנט? אם התשובה היא כן אפשר להגדיר יחס של ירושה.
- בכיוון ההפוך לא ניתן להגדיר יחס של ירושה לא כל אדם הוא גם סטודנט.
- שאלה: איזו מחלקת בסיס ניתן להגדיר לאדם?
- שאלה: איזו מחלקה יורשת ניתן להגדיר לדוקטורנט?
- שאלה: אילו תכונות משותפות יהיו לכל המחלקות?
- שאלה: אילו פעולות משותפות יהיו לכל המחלקות?



SYNTAX
שפה מייצרת חשיבה

המחלקה Person

```
public class Person {

   private String name;
   private int id;

   public Person(String name, int id) {
      this.name = name;
      this.id = id;
   }

   public void print(){
      System.out.println("This is a person with name " + name + " and id " + id);
   }

   public void sayPerson() {

      System.out.println("I am a person");
   }
}
```

המחלקה Student

```
public class Student extends Person {

    private String topic;

    public Student(String name, int id,String topic) {
        super(name,id);
        this.topic = topic;
    }

    @Override
    public void print() {
        super.print();
        System.out.println("And this is also a Student that studies " + topic);
    }

    public void sayStudent() {
        System.out.println("I am a student");
    }
}
```

כיוון שבמחלקת הבסיס הוגדר כבר constructor, אנו מפעילים אותו ע"י שימוש במילה שמורה super. נשים לב כי כל התכונות מוגדרות כפרטיות ולפיכך הגישה היחידה אליהן היא מתוך המחלקה שלהן.

כאן אנו עושים override לפונקציית ההדפסה של האב אבל בעזרת ה super אנו יכולים לקרוא לאותה פונקציה שכתבנו "עליה"

המחלקה Doctorant

```
public class Doctorant extends Student{

    private String thesis;

    public Doctorant(String name, int id, String topic,String thesis) {
        super(name,id,topic);
        this.thesis = thesis;
    }

    @Override
    public void print() {
        super.print();
        System.out.println("And he is also a Doctorant that doing thesis on " + thesis);
    }

    public void sayDoctorant() {
        System.out.println("I am a doctorant!!!");
    }
}
```

SYNTAX
שפה מייצרת חשיבה

Polymorphism פולימורפיזם

- פולימורפיזם - התייחסות אחידה לעצמים שונים .
- מקור השם מלטינית: פולי = הרבה, מורפיה = צורה.
- 3+5ו 3.4+5.2 פעולות אחרות בתוך המחשב אך הממשק נשאר זהה כנ"ל כאשר משרשרים מחרוזות.
- איך ניתן להחזיק את כלל אנשי האוניברסיטה במבנה אחד?
- הפולימורפיזם מאפשר להתייחס לעצמים שונים שיורשים מאותה מחלקת בסיס באופן אחיד.

הדגמה - פולימורפיזם

```
Person[] persons = new Person[3];
persons[0] = new Person("Moshe",1);
persons[1] = new Student("Yael",2,"computers");
persons[2] = new Doctorant("Yoni",3,"mechanics","newton");


for(int i=0;i<persons.length;i++) {
   persons[i].print(); // תיקרא הפונקציה המתאימה
}
```

SYNTAX
שפה מייצרת חשיבה

הסבר - למה זה עובד?

מדוע זה אפשרי?

<u>עיקרון 1</u> reference - בירושה reference :למחלקת בסיס יכול בפועל להצביע על עצם ממחלקה נגזרת. תכונה זו נקראת.down casting

<u>עיקרון 2</u> - פונקציות וירטואליות: כל פונקציה ב Java -מוגדרת כוירטואלית. זה אומר שמהדר דוחה את ההחלטה על ביצוע הקוד לזמן הריצה. בזמן הקומפילציה הוא יודע רק את כותרת הפונקציה אשר מוגדרת במחלקת הבסיס והוא מחזיק אותה בטבלה ללא המימוש. בזמן ריצה הוא יודע על מה מצביע ה reference ואז הוא יקרא למימוש הנכון.

אם העצם הוא מסוג Person תקרא פונקציהPerson.print()
אם העצם הוא מסוג Student תקרא פונקציהStudent.print()
אם העצם הוא מסוג Doctorant תקרא פונקציהDoctorant.print()

כל פונקציה ב Java מוגדרת כוירטואלית, לכן גרסת הפונקציה נקראת בהתאם לעצם (instance) ולא למצביע(reference)

- פונקציות וירטואליות - החלטה על גרסת הפונקציה נדחית מזמן ההידור לזמן הריצה של התכנית.
- מהדר משתמש בטכניקת) Late Binding קשירה מאוחרת.(
- לכל מחלקה מוגדרת טבלת גרסאות עדכניות לפונקציות.
- לכל עצם ממחלקה מוחזק מצביע לטבלה זו.
- בזמן ריצה נבחרת פונקציה מתאימה ע"י הסתכלות בטבלה.
- יש לשים לב, שיחס הרב צורתיות הוא חד כיווני.

Student s = new Person(...) // לא חוקי

פולימורפיזם - הרחבה בפונקציות

reference למחלקת בסיס יכול בפועל להצביע על עצם ממחלקה נגזרת.

יש לשים לב שההפך אינו נכון:

Reference למחלקה נגזרת לא יכול להצביע על עצם ממחלקת בסיס.

עברת פרמטרים לפונקציות

```
void f1 (Person p)
{

}
```

```
void f2 (Student s)
{

}
```

```
Person p1 = new Person("aaa",1);
Student s1 = new Student("bbb",2,"biology");

f1(s1); //OK!
f2(p1); //Compile error
```

SYNTAX
שפה מייצרת חשיבה

פולימורפיזם - casting

- במחלקת ה Student קיימת פונקציה שאינה קיימת במחלקת ה Person בשם sayStuednt.
- עם reference מסוג הבסיס, כיצד נוכל להפעיל את הפונקציה?
- הפתרון הוא **<u>בהמרה</u>** (casting)
- בהמרה לוקחים אחריות שבזמן ריצה הפונקציה תהיה קיימת.
- ההמרה מותרת ונחוצה רק כפי מעלה, כלומר למחלקה יורשת.
- במידה והעצם אינו מהמחלקה היורשת תיזרק שגיאה זמן ריצה ClassCastException

SYNTAX
שפה מייצרת חשיבה

Casting example

```
Person p = new Student("moshe",9044,"Biology");
p.sayStudent(); //שגיאת קומפילציה
double d = (double)5/2;
        ((Student)p).sayStudent(); //חוקי
```

SYNTAX
שפה מייצרת חשיבה

האופרטורinstanceof

- בג'אווה קיים בה מנגנון מיוחד בשם Runtime Type Information שמאפשר לקבל מידע על מצב התוכנית ועל העצמים בזמן ריצה .
- חלק מהמנגנון הוא האופרטור *instanceof.*
- האופרטור מקבל שני פרמטרים: עצם ומחלקה כלשהי בתוכנית, כך:

  <object> instanceof <class>

  ומחזיר ערך אמת באחד משלושת המקרים:

  העצם הוא מופע של המחלקה.
  העצם הוא מופע של מחלקה שירשה את המחלקה הנתונה.
  העצם הוא מופע של מחלקה שממשה את הממשק הנתון.

instanceof example

Person p = new Doctorant(“Moshik”,658797,”...”,”...”)

p instanceof Person → true

p instanceof Student → true

p instanceof Doctorant → true

Person p = new Person(“Afia”,7879000);

p instanceof Person → true

p instanceof Student → false

p instanceof Doctorant → false

מצא את השגיאה - instanceof

```
for(int i=0;i<persons.length;i++) {
   if(persons[i] instanceof Student) {
      ((Student) persons[i]).sayStudent();
   }
   else if(persons[i] instanceof Doctorant) {
      ((Doctorant)persons[i]).sayDoctorant();
   }
   else persons[i].sayPerson();
}
```

רמז, חישבו על
עקרונות
הפולימורפיזם....

Overriding Object functions

- למחלקה Object ממנה יורשים כל האובייקטים ב Java יש שתי פונקציות חשובות:

public String toString() // פונקציה המחזירה את מצב העצם

public boolean equals(Object other) // פונקציה הבודקת שוויון

- יתרונה של toString נעוץ באפשרות להעביר כל אובייקט לפונקציות ההדפסה ולקבל הדפסה אמיתית שלו.

public void println(Object o)

```
Person p = new Person(...)
System.out.println(p)
```

overriding equals

SYNTAX
שפה מייצרת חשיבה

- האופרטור == ו != עובדים על פרימיטיבים (על אובייקטים תעשה השוואה בין הכתובות בזכרון ולא השוואה בין התכנים).
- כדי לבצע השוואה אמיתית בין האובייקטים צריך לעשות override עלequals
- רבים ממבני הנתונים בשפה מאפשרים לשמור רשימה ללא חזרות על ידי קריאה ל equals של כל אובייקט .
- כך בעזרת הפולימורפיזם מתאפשר שימוש של כל אובייקט שנעשה בתוך ה API המובנה של השפה.
- להלן דוגמה ל override על: equals

SYNTAX
שפה מייצרת חשיבה

Equals example

```
public class Name {
  String firstName;
  String lastName;
  public boolean equals(Object o)
  {
    if (!(o instanceof Name)) return false;
    Name n = (Name)o;
    return firstName.equals(n.firstName)&&lastName.equals(n.lastName);
  }
}
```

גם המחלקה String עושה
overrideלequals

מניעת ירושה ומניעת דריסה

SYNTAX
שפה מייצרת חשיבה

```
final class Base {

}
```

לא חוקי - מחלקה final לא ניתנת לירושה

```
class Derived extends Base
{
```

המחלקה String מוגדרת כ final כדי למנוע שגיאות ולחסוך מעט במהירות

```
class Base {
    final void stam() {...}
}
```

לא חוקי - פונקציה final לא ניתנת לדריסה(override)

```
class Derived extends Base {
    void stam() {...}
}
```

```
final int num = 6;
```

לא חוקי - משתנה final לא ניתן לשינוי לאחר אתחולו

```
num = 11;
```

```
final Manger m  = new Manager();
```

לא חוקי - לשנות את ההצבעה

חוקי - השינוי הוא בתכונות האובייקט ולא במצביע שהוא final

```
m = null;
```

```
m.setSalery()
```