Java lecturer 11

אז על מה נדבר היום...

- Static
- ירושה
- פולימורפיזם - Polymorphism

RAVTECH

Static Class Members

ראינו שלכל מחלקה יש משתנים ומתודות ששיכים אליה.
הדרך שבה השתמשנו בהם היא על ידי יצירת מופע או שימוש פנימי.
בנוסף לכך כל מחלקה יכולה להכיל: משתנים סטטיים, מתודות סטטיות ואפילו מחלקות סטטיות.
הדרך להכריז על משתנים ומתודות סטטיות הוא על ידי ההכרזה **static** לפני ההכרזה על סוג המשתנה או במתודות לפני ההכרזה על סוג המתודה (void or type)

דוגמא למתודה סטטית מפורסמת:

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

גם תכונה של מחלקה יכולה להיות סטטית, לדוגמא:

```
static int number;
```

הדרך לגשת למשתנים והמתודות היא באמצעות שם המחלקה ולאחריה שם הפונקציה או המשתנה ללא יצירת מופע.
לדוגמא:

```
Double pi = Math.PI;

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

Static Class Members

משתנה סטטי

כאשר אנחנו מגדירים משתנה שהוא לא סטטי בתוך מחלקה, הוא למעשה נוצר מחדש בכל מופע של האובייקט שניצור.
לעיתים נרצה להגדיר משתנים אשר מוגדרים פעם אחת לכל האובייקטים מסוג מסוים.
לדוגמא:
אם נרצה ליצור משתנה שיספור כמה פעמים יצרנו מופע של אובייקט מסוים.

```
public class MyClass {

    static int numberOfInstance;

    public MyClass() {

        numberOfInstance++;
    }
}
```

רואים בעצם שלמשתנה יש רק מופע אחד בלבד והוא אינו שייך למופע, אלא משותף לכל המחלקה ולכל המשתמשים במחלקה.

הסיבה לכך שמשתנה סטטי נמצא **רק פעם אחת בזיכרון** והוא מאותחל בזמן יצירת במחלקה ולא ביצירת המופע.

Static Class Members

פונקציה סטטית

בדומה למשתנה סטטי, גם פונקציה סטטית היא פונקציה שמופעלת על המחלקה כולה ולא על מופע בודד של המחלקה.
לכן כדי לגשת לפונקציה סטטית אנחנו עושים זאת דרך שם המחלקה וציון הפונקציה (בניגוד לפונקציה לא-סטטית בה הגישה היא דרך שם המופע ואז שם הפונקציה).
פונקציות סטטיות שימושיות לנו עבור פעולות שנרצה לעשות מידי פעם ללא צורך בתלות במחלקה מסויימת.
מצד שני נרצה להשאיר את המתודות בתוך מחלקה שמייצגת את הנושא הכללי של הפונקציה.

לדוגמא:
אם נרצה ליצור פונקציות שעושות חישובים מתמטים.

```
public class MyMath {

  public static int add20(int num) {
   return num + 20;
  }
}
```

הדרך שבא נקרא לפונקציה היא:

```
int num = 200;
num = MyMath.add20(num);
System.out.println(num);
```

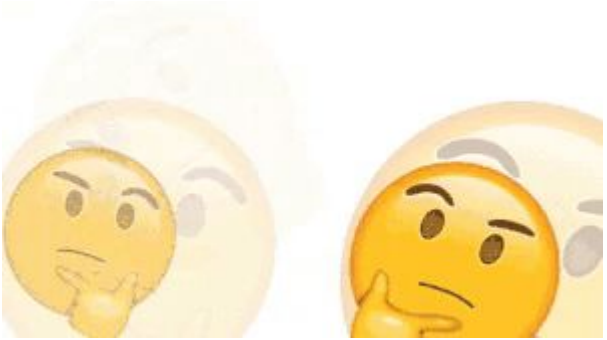
RAVTECH

Static Class Members

כללים לגבי חברי מחלקה סטטיים:

- חברי המחלקה הסטטיים– אינם שייכים למופע/אובייקט אלא למחלקה.
- הם נקראים משתנים של המחלקה.
- מכיוון שאינם משוייכים למופע – ניתן להשתמש בהם ללא יצירת מופע. תחת שם המחלקה.
- פונקציות סטטיות יכולות לגשת **רק** למשתני המחלקה הסטטים **ואינן יכולות** לגשת למשתני המופע.

שאלות?



Static Class Members

אז אחרי שראינו את הכללים לגבי חברי המחלקה הסטטים.
מה לא תקין בקוד הבא?

```
public class MyClass {

    Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        String str = scanner.next();
    }
}
```

- האם נוכל לשנות את המתודה main כך שלא תהיה סטאטית?
- מה נוכל לעשות כדי להגדיר את המשתנה ברמת המחלקה?
- האם הוא יהיה נגיש מכל המתודות?

RAVTECH

תרגול - Static Class Members

בדוק את המתודות הסטטיות תחת המחלקה Math.

- מצא את המתודה שמחשבת שורש והדפס את השורש של 64.

האם המתודות הללו משוייכות למופע? או שניתן להשתמש בהן באמצעות שם המחלקה?

צור מחלקה בשם Utils

חברי המחלקה:

- ערכו של המספר PI – הערך יהיה משותף לכל חברי המחלקה.
- Scanner scan משתנה סטטי שיהיה נגיש מכל התוכנית.

פעולות:

- getString – מתודה סטטית שקולטת String מהמשתמש ומחזירה אותו.
- getInteger – מתודה סטטית שקולטת int מהמשתמש ומחזירה אותו.
- מתודה סטטית שמקבלת קוטר של עיגול ומחזירה את ההיקף שלו

ירושה

ירושה היא כלי מאוד חשוב בתכנות מונחה העצמים.
הרעיון הוא בעצם מתן אפשרות להרחבה של תוכנית אחת בעזרת תוכנית אחרת היורשת ממנה את תכונותיה ומסוגלת להוסיף ולשנות אותן.
אז כמו שהזכרנו בהקדמה למחלקות שאנחנו רוצים לדמות את צורת התכנות לעולם שלנו באמצעות יצירת קטגוריות כלליות וקטגוריות משנה, (כמו חי צומח ודומם)
אז כאן בהורשה אנחנו ננסה כמה שיותר לבנות מחלקות שמיצגות משהו מרכזי ומהם מחלקות אחרות יירשו את התכונות והמחלקות האלו יוסיפו עוד תכונות ופעולות ספציפים.
לדוגמא ניקח למשל כלב. שאפשר לשייך אותו לקטגוריות רבות

- בעל חיים
  - יונק
    - יונק טורף
- משפחת הכלביים
  - זאב מצוי
    - כלב הבית
- רטריבר
  - לברדור

בהורשה יש לנו את האפשרות לבטא את הרעיון הזה בקוד מחשב.

ירושה

אז איך זה קורה?
ירושה בין מחלקות היא באמצעות המילה **.extends**
ניתן כל פעם לרשת רק ממחלקה אחת בלבד. (בשונה מ Python או C++)
בשביל הדוגמא נקח מחלקה שאנחנו מכירים.

```
public class Person {

   private String firstName;
   private String lastName;
  ...
}
```

עכשיו ניצור מחלקה חדשה בשם Student שהיא תירש מ Person

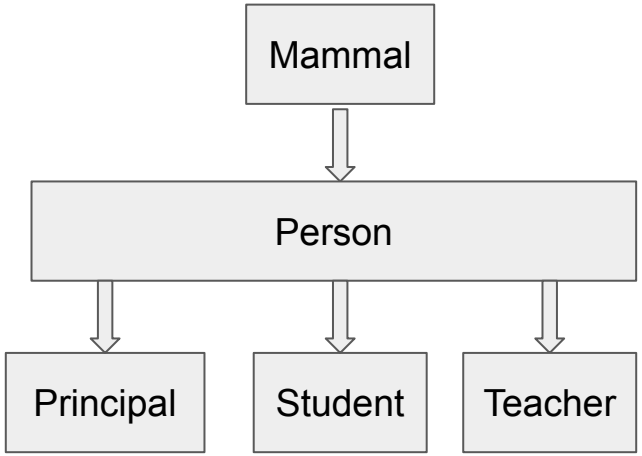
```
public class Student extends Person {

   private boolean isPay;
   private double GPA;
}
```

אפשר בעצם להגיד שכל תלמיד הוא קודם כל בן אדם, וכך גם מורה ומנהל.
רק שלתלמיד בנוסף לזה שהוא בן אדם יש לו תכונות נוספות שיש אותם רק לתלמיד ולא למישהו אחר.

ירושה

מה שיצרנו כאן הוא בעצם סוג של היררכייה בין המחלקות שניתן לדמות אותה לסוג של פירמידה.
במיוחד אם נרצה לדמות אותה לעולם האמיתי שראינו בדוגמא של הכלבים, שזה נראה כך:



אנחנו רואים שאנחנו יכולים ליצור היררכיה של מחלקות עבור מורה, תלמיד ומנהל, וכולם ירשו ממחלקה של אדם שגם היא תירש ממחלקה של יונקים.

ירושה

אז אחרי שהגדרנו את התכונות גם במחלקת האב וגם במחלקת הבן, נרצה שלמחלקת הבן תהיה האפשרות לגשת לתכונות של מחלקת האב.
בשביל זה נצטרך 2 דברים.
נכיר דבר ראשון את המילה השמורה **super.**
בדומה למה שעד היום השתמשנו במילה **this** לצורך פנייה לחלקי המחלקה השונים, על ידי השימוש במילה **super** נוכל לפנות גם לחלקי מחלקת האב השונים.

לדוגמא:

```
public class Student extends Person {

   private boolean isPay;
   private double GPA;

   public Student(boolean isPay, double GPA, String firstName) {
       this.isPay = isPay;
       this.GPA = GPA;
       super.firstName = firstName;

   }
}
```

RAVTECH

ירושה

דוגמא נוספת לשימוש במילה **super**:

שימו לב שבמקרה כזה, אם אין למחלקת האב default constructor.
נהיה **חייבים** לממש את הבנאי של מחלקת האב בכל הבנאים של המחלקה היורשת **לפני** שעושים השמה על התכונות של המחלקה היורשת.

```
public class Person {

   private String firstName;
   private String lastName;

   public Person(String firstName, String lastName) {
       this.firstName = firstName;
       this.lastName = lastName;
   }

}

public class Student extends Person {
...
   public Student(boolean isPay, double GPA, String firstName,String lastName) {
       super(firstName,lastName);
       this.isPay = isPay;
       this.GPA = GPA;
   }
}
```

RAVTECH

ירושה

הדבר הבא שנצטרך זה דבר שהזכרנו אותו בעבר.
מה הבעיה בקטע הקוד הבא?

מכיון שהתכונות מוגדרות כ **private** אין לנו אפשרות לגשת אליהם.
הדרך לגשת אליהם מבלי להפוך אותם **public** היא באמצעות הרשאת הגישה **protected** .
וזה נראה כך:

```
public class Person {

   protected String firstName;
   protected String lastName;

}

public class Student extends Person {
...
   public Student(boolean isPay, double GPA, String firstName,String lastName) {
       super.firstName = firstName;
       super.lastName = lastName;
       this.isPay = isPay;
       this.GPA = GPA;
   }
}
```

RAVTECH

ירושה

קריאה לפונקציות ממחלקת האב.
למחלקת הבן (המחלקה היורשת) ישנם את אותו היכולות שיש למחלקת האב.
לדוגמא

```
public class Person {

   ...
      public void printDetails() {

            System.out.println(String.format("First name: %s Last name: %s", firstName, lastName));
      }
}

public class Student extends Person {
   ...
}

public static void main(String[] args) {

    Student student = new Student();
    student.printDetails();
}
```

RAVTECH

ירושה

Override

באותה הדרך שבה אנו קוראים לתכונות, אנחנו יכולים גם לקרוא למתודות ממחלקת האב במחלקה היורשת.
וכן גם אנחנו יכולים "לדרוס" (Override) פונקציות עם אותו השם.
לדוגמא :

```
public class Person {

   ...
      public void printDetails() {

            System.out.println(String.format("First name: %s Last name: %s", firstName, lastName));
      }
}

public class Student extends Person {
...
   public void printDetails() {

       super.printDetails();
       System.out.println(String.format("GPA: %f ", GPA));
   }
}
```

RAVTECH

פולימורפיזם

פולימורפיזם - רב צורתיות
בפולימורפיזם אנחנו מתייחסים לעצמים שונים בתור דברים דומים.
למשל: מחשב נייח, מחשב נייד, שרת וטלפון חכם כולם יכולים לירוש מאובייקט שנקרא מחשב.
אמנם ביום יום נוכל לקרוא לכולם מחשבים סתם.
אבל אם למשל אנחנו אומרים שאנחנו רוצים לכבות אותם.
אז כדי שנדע איך, אנחנו צריכים לדעת על איזה מחשב מדובר ולכן אנחנו נגדיר כל אחד בנפרד.

נסתכל על הדוגמאות שאנחנו מכירים:
הגדרנו מחלקה בסיסית מסוג person,
עכשיו נרצה להוסיף עוד שלושה מחלקות, Student, Employee ו - Teacher.
נניח שנרצה ליצור מערך, שיכיל את כל הישויות שקיימות בבית הספר.
במקרה הזה לא נוכל לבנות מערך של סטודנטים, עובדים או מורים, מכיוון ששתי הישויות האחרות לא תוכלנה להיות חלק מהמערך.
אבל, אם נבנה מערך מסוג person, נוכל להשתמש בעיקרון הפולימורפיזם:
כל מצביע על עצם בסיס יכול להצביע בהתאם גם על עצם של מחלקה שנגזרה ממחלקת הבסיס.
כלומר, הפונקציה תקרא לעצם לפי מה שהוא באמת ולא תלך לפונקציות של מחלקת הבסיס.
במקרה הזה נגדיר מערך מסוג person אבל העצמים שנציב יהיו מסוגים שונים.

פולימורפיזם

וזה נראה כך:

```
ArrayList<Person> people = new ArrayList<>();

Student student = new Student();
Teacher teacher = new Teacher();
Employee employee = new Employee();

people.add(student);
people.add(teacher);
people.add(employee);
```

עכשיו כשנקרא לפונקצית ההדפסה יודפסו פונקציות ההדפסה המתאימות לכל מחלקה ולא פונקצית ההדפסה של person.

```
for (Person person : people) {

   person.printDetails();
}
```

RAVTECH

פולימורפיזם

אז איך זה עובד?

המחשב בעצם דוחה את ההחלטה על גירסת הפונקציה מזמן ההידור לזמן הריצה וכך הוא יכול להחליט בזמן הריצה לאיזה גירסה לקרוא, בהתאם לסוג העצם שאליו מתייחס המצביע.

שימו לב שהפולימורפיזם הוא חד כיווני.
כלומר, אי אפשר להתייחס לעצם שמוגדר עפ"י מחלקת הבסיס בעזרת מצביע לעצם של מחלקה שנגזרה ממנו.
לדוגמא:

```
Student student1 = new Person();
```

פולימורפיזם

דוגמא נוספת

```
static void printPersonDetails(Person person){

   person.printDetails();
}

public static void main(String[] args) {

   Student student = new Student();
   Teacher teacher = new Teacher();

   printPersonDetails(student);
   printPersonDetails(teacher);
}
```

הקריאה האחרונה למתודה *printPersonDetails* היא חוקית, כיוון ש Student הוא סוג של Person.
ומתוך הקוד של printPersonDetails תיקרא המתודה המתאימה של Student , ויודפס הטקסט המתאים עבור Student .
וכל זאת, על אף שבמתודה זאת המשתנה person מוגדר כמשתנה מטיפוס Personולא כמשתנה מטיפוס Student.

כללי תורשה

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 את אותן פונקציות.
- תמיד בקריאה לפונקציה תיבחר הגירסה העדכנית ביותר שלה.
- אין אפשרות לרשת מכמה מחלקות שונות.

תרגילים