Java lecturer 9

אז על מה נדבר היום...

- Varargs - מספר משתנה של פרמטרים
- Encapsulation - כימוס
- Modifiers - הרשאות גישה
- מתודות סטטיות

RAVTECH

Varargs

עד היום ראינו שמתודה יכולה לקבל רק את מספר הפרמטרים שהגדרנו מראש.
לדוגמא:

```
public void getArgs(String arg1, String arg2) {

    System.out.println(arg1 + " " + arg2);
}
```

יכול להיות שנרצה לבנות פונקציה שתקבל מספר לא ידוע של פרמטרים.
בשביל זה נשתמש ב 3 נקודות ...

```
public void getArgs(String arg1, String... arg2) {

    for (int i = 0; i < arg2.length; i++) {

        System.out.println(arg1 + " " + arg2[i]);
    }
}
```

וככה נוכל לקרוא לפונקציה בדרך הבאה:

```
getArgs("Hello: ","Ted","Marshall","Robin","Barney");
```

כלל - Varargs

הפרמטר המשתנה **חייב** להיות האחרון ברשימת הפרמטרים של המתודה.
וזה בשביל שהקומפיילר ידע לשייך את כל המשתנים האחרונים ל-varargs.

הפרמטר האחרון שהמתודה מקבלת

```
public void getArgs(String arg1, String... arg2) {

    for (int i = 0; i < arg2.length; i++) {

        System.out.println(arg1 + " " + arg2[i]);
    }
}
```

RAVTECH

Varargs תרגול

במחלקה Book כתבנו פונקציה שמחפשת מתכון לפי רכיב.
נכתוב עוד פונקציה שמחפשת מתכון שמכיל מספר מרכיבים.
הפונקציה תקבל מספר לא מוגבל של רכיבים ותחזיר רק את המתכונים שמכילים את **כל** הרכיבים האלו:
לדוגמא:
אם יש לנו מתכון עבור בצק לפיצה שמכיל את המרכיבים הבאים:
קמח, שמרים, שמן, סוכר, מלח, מים
ויש גם מתכון עבור רוטב לפיצה שמכיל:
עגבניות, בצל, שום, שמן, סוכר, פלפל שחור, מלח
אז עבור חיפוש הרכיבים שמן, סוכר ומלח נקבל את 2 המתכונים האלה
אבל עבור חיפוש שמן, סוכר ,מלח וקמח נקבל רק את המתכון של הבצק.

תרגול Varargs

פתרון:

במחלקה Book

```
public void printBySeveralIngredient(String... ingredient) {

   for (Recipe recipe : recipes) {

      if (recipe.isContainIngredients(ingredient))
           System.out.println(recipe.toString());
   }
}
```

במחלקה Recipe

```
public boolean isContainIngredients(String[] ingredient) {
   ArrayList<String> isContainsArray = new ArrayList<>();
   for (Ingredient ingredient1 : ingredients)
       for (String s : ingredient)
           if (ingredient1.getName().equals(s))
               isContainsArray.add(s);

   return isContainsArray.size() == ingredient.length;
}
```

RAVTECH

Encapsulation - כימוס

כימוס או הסתרה הינו אחד מהעקרונות החשובים בתכנות מונחה עצמים.
תכנות מונחה עצמים מאפשר כימוס:
כימוס הכוונה מניעת שימוש בחלק או ברוב חלקי המחלקה עבור צרכים שונים.

יתרונות:
מפתחים שמשתמשים במחלקה, יקבלו גישה רק לשירותים שהמחלקה מספקת על ידי מתודות ספציפיות בלבד.
אין למשתמשי המחלקה גישה או ידע על המבנה הפנימי של המחלקה.
אין למשתמשי המחלקה גישה או ידע לגבי המשתנים הפרטיים של המחלקה.

התכונות הקודמות שצוינו מאפשרות מודולריות וגמישות:
לדוגמא ניתן לשנות את המבנה הפנימי של המחלקה (תחזוקה ושיפורים). מבלי לשנות קוד שמשתמשי המחלקה כתבו.

עבור דוגמא נכתוב מחלקה נוספת עבור התוכנה שבנינו.
עד עכשיו יצרנו רק ספר אחד, מה אם נרצה שיהיה לנו מספר ספרים של מתכונים.
לצורך כך נכתוב מחלקה שנקראת Library שהיא תכיל רשימה של ספרים.

Encapsulation - כימוס

```
public class Library {
   ArrayList<Book> books = new ArrayList<>();

   void addBook(Book book) {

      books.add(book);
   }

   public Book findBookByName(String name) {
       for (int i = 0; i < books.size(); i++)
           if(books.get(i).name.equals(name))
               return books.get(i);

       return null;
   }

}
```

יצרנו מחלקה בשם ספריה שהיא מאחסנת את כל הספרים.
ועכשיו מפתח אחר שלא מכיר את מה שכתבנו רוצה להשתמש במחלקה הזאת על מנת להוסיף או למצוא ספרים.
מה הבעיה העיקרית של התוכנה שלנו?
שהיא מוגבלת עד 5 ספרים בלבד
מה נעשה כדי לפתור את זה.
נשנה את מערך הספרים ל ArrayList.

מה הבעיה במקרה כזה?
שאם המפתח השני ניגש באופן ישיר לפרמטר **books** בצורה הבאה:

```
Book[] books = library.books1;
```

הוא מיד יקבל שגיאה
ואת זה אנחנו רוצים למנוע על ידי כימוס שיתן לנו במקרה הזה מודולריות וגמישות לשנות את המבנה הפנימי של המחלקה מבלי לשנות את המתודות החשופות לעיני חוץ.

Encapsulation - כימוס

אז כמו שראינו, כימוס מעניק לנו את היכולת להשתמש בקוד מקור מבלי לדאוג לפרטי המימוש שלו.
כימוס מאפשר התמחות מפתחים:
מפתח אחד מומחה לגרפיקה תלת-מימדית, מפתח אחר כותב משחקים ושניהם יכולים להשתמש באותו הקוד מבלי לדאוג לפרטי המימוש.
כתיבה נכונה תאפשר הוספה או עדכון של מידע אך ורק דרך מתודות.
כך ניתן לבצע בדיקות ולידציה על הערכים הנכנסים ולמנוע שינויי קוד עתידיים.
לדוגמא מהתוכנה שכתבנו:
אם נרצה להגביל את משתנה ה - rate של המתכון שלא יהיה יותר גבוה מ - 5 ולא יהיה יותר נמוך מ -0
אז נכתוב את המתודה הבאה במחלקה Recipe

```
public void setRate(int rate) {
    if (rate > 5)
        this.rate = 5;
    else if (rate < 0) {
        this.rate = 0;
    } else {
        this.rate = rate;
    }
}
```

עכשיו מפתח שירצה לשנות את הדירוג יוכל להכניס כל מספר אבל המחלקה בעצמה "תדאג" לזה שהדירוג ישאר תמיד בין 0 ל- 5.

*מה הבעיה שיכולה להיות לנו?
על זה בשקף הבא.

RAVTECH

Modifiers - הרשאות גישה

```
public class Book {

   private String name;

}

public static void main(String[] args) {

   Book book = new Book();

   book.name = "Shabbat recipes";

}

public void setName(String name) {
   this.name = name;
}
```

שימו לב לקטע הקוד הבא:

מה יקרה עכשיו?
במקרה הזה תקרה שגיאת קומפילציה ולא נוכל להריץ את התוכנית.
מכיון ששינינו את הרשאת הגישה לשדה **name** על ידי כך שבמקום **public** הגדרנו אותו **private**.
שזה אומר שהמשתנה לא ניתן לשינוי מחוץ למחלקה.

אם בכל זאת נרצה לאפשר שינוי איך נעשה את זה?
נכתוב מתודה שנקרא לה set בתוך המחלקה שהיא תהיה אחראית על השינוי.

Modifiers - הרשאות גישה

ראינו אם כן שמשתנים שמוגדרים כ- private נגישים אך ורק מתוך המחלקה.
שזה הנושא העיקרי של כימוס שנותן לתוכנה שלנו את היתרונות שהזכרנו מקודם על כימוס וככה אנחנו מקבלים שליטה על קוד המקור מצד אחד ומצד שני נותנים את האפשרות להשתמש בקוד שכתבנו.
לסיכום:

- המחלקה לא חושפת את המשתנים.
- המחלקה יכולה לאפשר שינוי ערכים ע"י מתודות שנועדו לכך.
- מחלקות אחרות יכולות "לבקש" מהמחלקה לשנות את המידע שלה.
- משתנים שמוגדרים כ-public לעומת זאת, נגישים גם מתוך המחלקה וגם מחוץ למחלקה.

אז כמו שראינו לגבי משתנים, אותו הדבר חל גם עבור מתודות ובנאים.
מתודה פרטית נגישה אך ורק מתוך המחלקה.

Modifiers - הרשאות גישה

בנוסף ל 2 סוגי ההרשאות ישנם עוד 2 הרשאות נוספות:

1. **`private`** נגיש אך ורק מתוך המחלקה.
2. **`public`** נגיש מתוך המחלקה ומחוצה לה.
3. default-access נגיש לכל המחלקות שנמצאות באותו package.
4. **`Protected`** נגיש רק בירושה. ועל זה נלמד בהמשך.

RAVTECH

הרשאות גישה - Modifiers

כשאנחנו לא מיצרים מופע חדש ואנחנו רק עושים השמה יכול לקרות לנו שגיאות.
בואו נסתכל על קטע הקוד הבא:

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

    Book javaFund = new Book("Java Fundamentals");
    Library library = new Library();
    library.addBook(javaFund);

    ArrayList<Book> books = library.getBooks();
    books.clear();
    books.add(new Book("Hacked!"));

    System.out.println(library.getBooks().get(0).getName());
    Book b = library.findBookByName("Hacked!");
    b.setName("Youve been hacked...Again.");
    System.out.println(b.getName());

}
```

הבעיה בקטע הקוד הזה היא שאובייקטים לא מעבירים את הערך שלהם בהשמה.
אובייקטים מועברים לפי כתובת בזיכרון!
וברגע שאפשרנו למחלקה אחרת לגשת לכתובת הזאת המידע נגיש והיא תוכל גם לשנות אותו.

RAVTECH

בדיקת שוויון לפי ערך ולפי מצביע

נסתכל על דוגמא נוספת שתראה לנו את זה באופן מוחשי יותר:

מה יודפס בשלושת הפעמים?

כל זה מכיוון שאובייקטים מועברים לפי כתובת בזיכרון!
ואם הכתובת היא שונה הם לא אותו הדבר.

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

    Book javaFund = new Book("Java Fundamentals");
    Book javaFund2 = new Book("Java Fundamentals");
    int i = 1;
    int j = 1;

    System.out.println(javaFund == javaFund2);  → false

    System.out.println(i == j);  → true

    Point p1 = new Point(1, 2);
    Point p2 = p1;
    p2.x = 3;
    System.out.println(p1.x);  → 3

}
```

RAVTECH

תרגול יצירת עותק

לפעמים נרצה לתת גישה לאובייקט עצמו ולפעמים נרצה רק לתת את הערכים של האובייקט בלבד.
בשביל זה נצטרך ליצור עותק של האובייקט ואותו להחזיר.
יש ליצור במחלקה Book מתודה שמעתיקה את האובייקט לאובייקט חדש ומחזירה את העותק.

למחלקה יהיו את היכולות הבאים:

- תכונות:
  - מחבר
  - כותרת
  - תקציר
- בנאים:
  - בנאי שמקבל את כל התכונות.
  - בנאי שמחזיר אובייקט מועתק.
- פעולות:
  - Getters and Setters לכל התכונות
  - החזר עותק

בדיקת שוויון לפי ערך ולפי מצביע

ראינו שאובייקטים מועברים לפי כתובתם בזכרון.
להבדיל ממשתנים פרימטיביים שמועברים לפי ערך.
ולכן בדיקת שוויון תמיד תחזיר false אם מדובר בשני מופעים שונים.
אפילו אם כל התכונות של האוייבקטים זהות.
הסיבה: הם אינם מצביעים לאותו מקום בזכרון ולכן הם אינם אותו אובייקט.

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

    Book javaFund = new Book("Java");
    Book javaFund2 = new Book("Java");
    System.out.println(javaFund == javaFund2);  false

}
```

כדי לבדוק שוויון לפי ערך עבור אובייקטים עלינו לכתוב מתודה שבודקת שוויון לפי ערך.
למזלנו... קל מאוד לייצר את המתודה הזו בסביבת העבודה:
על ידי צמד הכפתורים Alt + Insert
ובוחרים בתפריט generate hashcode and equals מסמנים את הערכים שברצוננו לבדוק ולוחצים על Next
המתודה equals שנוצרת עבורנו מסייעת בבדיקת שוויון לפי ערך.

RAVTECH

בדיקת שוויון לפי ערך ולפי מצביע

עכשיו נוכל לבדוק שוויון לפי ערך על ידי המתודה שייצרנו:

```
Book b1 = new Book("title");
Book b2 = new Book("title");
System.out.println(b1.equals(b2));//true> equal by value.
```