Java lecturer 13

אז על מה נדבר היום...

- Interface

Interface

לצורך ההבנה של Interface נתחיל עם דוגמא קטנה.
יש לנו 2 מחלקות, אחת מייצגת תלמיד ואחת מייצגת מרצה.
במקרה הזה **אין להם אב משותף**.
אבל לשניהם יש מתודה שמאפשרת להם להשאיל ספר מהספרייה.

```
public class Lecturer {
  ...
   public void showPresentation() {/*...*/}

   public void borrowBook(Book book) {/*...*/}
}

public class Student {
  ...
   public void doHomework() {/*...*/}

   public void borrowBook(Book book) {/*...*/}
}
```

Interface

עכשיו נרצה לבנות מחלקה שמייצגת ספריה.
למחלקה יהיה 2 תכונות, רשימת ספרים ורשימת מנויים ומתודה להשאלת ספרים.

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

   public void borrowBook(Object o, Book bookName) {
       if (o instanceof Lecturer)
           ((Lecturer) o).borrowBook(bookName);
       else if (o instanceof Student)
           ((Student) o).borrowBook(bookName);
   }
}
```

מה תהיה הבעיה שלנו במקרה הזה?
שמכיון שחוץ מזה שלכל מי שירצה להשאיל ספר צריך שיהיה לא את היכולת להשאיל, נצטרך במחלקה של הספריה לבדוק כל אחד בנפרד.
ואם מישהו הוסיף את היכולת במחלקה כלשהיא אבל לא עדכן את הקוד של הספריה עדיין הוא לא יוכל להשאיל.

Interface

אז אחרי שהבנו את הבעיה נוכל להסביר איך אנחנו פותרים אותה.
Interface - ממשק (מנשק)
הממשק הוא בעצם סוג של מחלקה אבסטרקטית **אבל** ללא תכונות וללא שיטות שאינם אבסטרקטיות וכמובן שגם ללא בנאי.
כל השיטות בממשק הם אבסטרקטיות וללא מימושים.
המטרה בזה היא עבור "הכרחה" של מימוש אוסף יכולות בעלי מכנה משותף מצד אחד ומצד שני להימנע מירושה.
לדוגמא:
גם Animal וגם Shape יכולים להדפיס את עצמם אבל לא הגיוני ששניהם יירשו ממחלקה של Print
אז מכיון שנרצה רק להוסיף יכולות ולא לרשת בדיוק בשביל זה קיים הממשק.
ומעכשיו גם Animal וגם Shape **יממשו** ממשק משותף עם הוראת הדפסה.

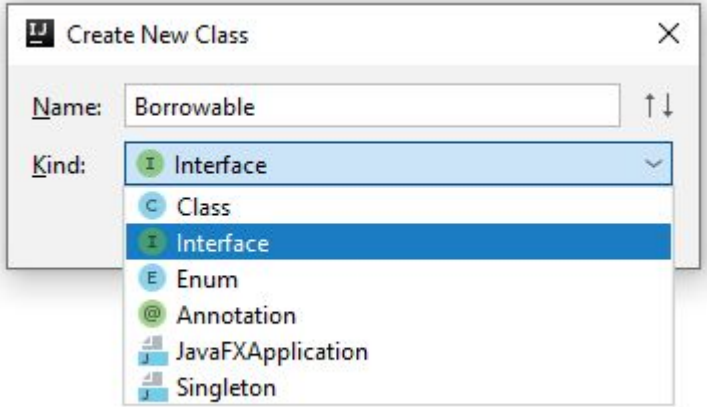
אז לצורך הפתרון של הבעיה שלנו גם Lecturer וגם Student יממשו את הממשק Borrowable ש"יכריח" אותם להוסיף את המתודה borrowBook.

אז איך זה נראה?
אם עד היום שיצרנו מחלקות קראנו למחלקה בשם **class** אז לצורך מימוש ממשק ניצור מחלקה חדשה ונקרא לה **interface**.
הקוד שנקבל ייראה כך:

```
public interface Borrowable {
}
```

Interface

איך נעשה את זה בתוכנה שלנו?
די פשוט...
ניצור מחלקה חדשה אבל בסוג המחלקה נבחר ב Interface.



Interface

נוסיף לממשק שיצרנו את הפונקציה borrowBook.

```
public interface Borrowable {
   void borrowBook(Book book);
}
```

אחרי שיצרנו ממשק כזה נממש את היכולת הזאת אצל Lecturer ו Student.
אופן המימוש הוא באמצעות המילה **implements** וזה יראה כך:

```
public class Student implements Borrowable {
...
   @Override
   public void borrowBook(Book book) {}

}
```

אותו הדבר נעשה גם למרצה:

```
public class Lecturer implements Borrowable {
...
   @Override
   public void borrowBook(Book book) {}

}
```

RAVTECH

Interface

אחרי שהוספנו למרצה ותלמיד את היכולת להשאיל ספרים נרצה לשדרג גם את הקוד של הספריה שלנו.

עכשיו נוכל לשנות אותו שיראה כך:

```
public class Library {

    private ArrayList<Book> books = new ArrayList<>();
    private ArrayList<Borrowable> subscribers = new ArrayList<>();

    public void borrowBook(Borrowable o, Book bookName) {
        o.borrowBook(bookName);
    }

}
```

Interface

נקודות חשובות

1. מכיוון שממשק מציין יכולות מוסכמה שהשם שלו יסתיים ב- able
   למשל: Borrowable, Printable
2. שימו לב שמחלקות יכולות לממש הרבה ממשקים במקביל! בשונה מירושה.
3. מכיוון שאנחנו מגדירים את המחלקה כ- interface ולא כ- class,
   אין צורך לציין שהשיטות הן public או abstract, הקומפיילר יודע להתייחס אליהן כך אוטומטית
4. כשאנחנו ממשים ממשק וגם יורשים ממחלקה, הממשק יבוא לאחר המחלקה.
   לדוגמא:

```
public class Student extends Person implements Borrowable
```

RAVTECH

Interface תרגול

מכיוון שלא כל החיות יכולות להשמיע קול, לכן לא נרצה לאפשר פעולה זו עם מימוש ריק במחלקת הבסיס Animal
ולפי מה שאמרנו, השמעת קולות זה רק תכונה התנהגותית.
לכן נכתוב ממשק המכיל את הפעולה getNoise.
כל החיות ירשו מ- Animal וחיות שיודעות להשמיע קול יממשו את הממשק Noiseable
בתוכנית נדפיס את כל החיות ובמקרה שהחיה יודעת להשמיע קול, נדפיס גם את הקול שהם עושות.
בנוסף נגדיר גם לכל חיה שם ואת הפונקציה של getName נגדיר כסופית כדי שלא תהיה ניתנת לדריסה.

Arrays.sort

שימוש בממשקים קיימים

המתודה Arrays.sort מקבלת מערך וממיינת את איבריו באמצעות השיטה Bubble Sort

עבור כל הטיפוסים הבסיסיים המתודה יודעת לבצע את העבודה מאחר והיחס של המשתנה והפעולה > עובדים טוב.

לדוגמא:

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

    int[] arr = {3, 7, 1, 9, 2, 6};
    System.out.println("before sort: " + Arrays.toString(arr));
    Arrays.sort(arr);
    System.out.println("after sort:  " + Arrays.toString(arr));

}
```

Output:

before sort: [3, 7, 1, 9, 2, 6]
after sort: [1, 2, 3, 6, 7, 9]

Arrays.sort

עכשיו ננסה לעשות את אותו הדבר על מערך של אובייקטים שאנחנו בנינו:

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

    Person[] arr = new Person[3];
    arr[0] = new Person("Moshe", 18);
    arr[1] = new Person("David", 15);
    arr[2] = new Person("Israel", 27);

    System.out.println("Persons are: " + Arrays.toString(arr));
    Arrays.sort(arr);
}
```

במקרה הזה אנחנו נקבל את הפלט והשגיאה הבאה:

```
Persons are: [Person{Name='Moshe', age=18}, Person{Name='David', age=15}, Person{Name='Israel', age=27}]

Exception in thread "main" java.lang.ClassCastException: class Person cannot be cast to class java.lang.Comparable
        at java.base/java.util.ComparableTimSort.countRunAndMakeAscending(ComparableTimSort.java:320)
        at java.base/java.util.ComparableTimSort.sort(ComparableTimSort.java:188)
        at java.base/java.util.Arrays.sort(Arrays.java:1250)
        at MyClass.main(MyClass.java:25)
```

RAVTECH

Arrays.sort

כדי להבין את הסיבה לשגיאה שראינו מקודם, נצטרך לנסות לממש את זה בעצמנו.

אם ניתן כפרמטר מערך של אובייקטים הקומפיילר לא ידע מה הפירוש של האופרטור > עבור Object..

```
public static void bubbleSort(Object[] arr) {
    for (int i=arr.length-1 ; i > 0 ; i--)
        for (int j=0 ; j < i ; j++)
            if (arr[j] < arr[j+1]) {
                Object temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

במקרה שלנו המתודה sort רוצה שהאיברים למיון ידעו לתת תוצאת יחס בין שני איברים (גדול/קטן/שווה)
אבל כיוון שראינו שלא כל הטיפוסים ברי השוואה באמצעות האופרטור >
לכן השיטה מתבסס על כך שהם יממשו את ההתנהגות המוגדרת בממשק הקיים Comparable המכיל את השיטה compareTo.
ואם נכנס לעומק הקוד נראה את המתודה בשקף הבא:

Arrays.sort

כך נראית הפעולה:

```
public static void bubbleSort(Comparable[] arr) {
    for (int i=arr.length-1 ; i > 0 ; i--) {
        for (int j=0 ; j < i ; j++) {
            if (arr[j].compareTo(arr[j+1]) > 0) {
                Comparable temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
}
```

ושימו לב גם לשם השגיאה שהתקבלה כשניסינו לרוץ על המערך:

Exception in thread "main" java.lang.ClassCastException: class Person cannot be cast to class java.lang.Comparable

Arrays.sort

אז כדי שגם המחלקה שלנו תתמוך ביחס ההשוואה בין האובייקטים, נצטרך לממש את המתודה compareTo.
וכדי לעשות את זה נממש את הממשק הקיים Comparable
הפונקציה compareTo עובדת בצורה מאוד פשוטה על ידי החזרת הפרמטרים הבאים

- 0 במקרה ששני האובייקטים זהים
- 1 במקרה שאובייקט א' גדול מאובייקט ב'
- -1 במקרה שאובייקט א' קטן מאובייקט ב'

אז אם נרצה לסדר את כל האנשים לפי הגיל, הפונקציה שנכתוב תראה כך:

```
public class Person implements Comparable<Person> {
...
   @Override
   public int compareTo(Person p) {
       if (age < p.age)
           return -1;
       else if (age > p.age)
           return 1;
       return 0;
   }
}
```

Arrays.sort

עכשיו נוכל לכתוב את הפונקציה שלנו למיון מערך של אנשים:

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

    Person[] arr = new Person[3];
    arr[0] = new Person("Moshe", 18);
    arr[1] = new Person("David", 15);
    arr[2] = new Person("Israel", 13);

    System.out.println("Persons are: " + Arrays.toString(arr));
    Arrays.sort(arr);
    System.out.println("Persons are: " + Arrays.toString(arr));

}
```

Output:
Persons are: [Person{Name='Moshe', age=18}, Person{Name='David', age=15}, Person{Name='Israel', age=13}]
Persons are: [Person{Name='Israel', age=13}, Person{Name='David', age=15}, Person{Name='Moshe', age=18}]

Arrays.sort

תוספת קטנה.
מה נעשה במקרה שנרצה לסדר לפי שמות?
אז בשביל זה מחלקת String מממשת כבר את הפונקציה compareTo וכל מה שנצרך לעשות זה די פשוט.

```
public class Person implements Comparable<Person> {
...
   @Override
   public int compareTo(Person p) {

      return name.compareTo(p.name);

   }
}
```

תרגילים

יש ליצור את המחלקות הבאות:

- גנב
- אצן
- צ'יטה

לכל המחלקות יש ליצור בנאי, get and set וכולם יורשות מאבות המתאימים להם שייצרנו בעבר

לכל גנב בנוסף לפרטיו הבסיסים נוסיף את גובהו ואת מספר הפעמים שנעצר. וכן נוסיף את המתודות steal ו- run
לכל צ'יטה נוסיף גם את שמה, גובהה, מספר החיות שטרפה, ונספק לה את המתודה run
לכל אצן נוסיף גם את מהירות הריצה הממוצעת שלו, ונממש עבורו המתודות run ו- breath
יש לאפשר למיין את האצנים עפ"י שמם
יש לאפשר למיין את הגנבים עפ"י מספר הפעמים שנעצרו

נכתוב תוכנית:

נשאל את המשתמש כמה אובייקטים לייצר במערך
עבור כל אובייקט שאלו את המשתמש מאיזה טיפוס האובייקט
הדפיסו את כל איברי המערך, ועבור כל אובייקט שיכול, יש להפעיל עבורו את השיטה steal ועבור כל אובייקט שיכול להפעיל את השיטה run
הגדירו מערך של אצנים והציגו אותם ממויינים לפי שמם
הגדירו מערך של גנבים, מיינו אותם לפי מספר הפעמים שנעצרו והציגו את המערך הממוין