מחלקות ושרותים מוכללים
Java Generics

אוהד ברזילי
תכנות מתקדם בשפת Java
אוניברסיטת תל אביב

מצגת זו מבוססת על מצגות של פרופ' עמירם יהודאי ופרופ' סיוון טולדו מאוניברסיטת תל אביב

מבנים מקושרים

- כדי לייצג מבנים מקושרים, כגון רשימה מקושרת, עץ, וכדומה, מגדירים מחלקות שכוללות שדות שמתייחסים לעצמים נוספים מאותה מחלקה (ולפעמים גם למחלקות נוספות).
- כדוגמא פשוטה ביותר, נגדיר מחלקה `IntCell` שעצמים בה מייצגים אברים ברשימות מקושרות של שלמים.
- המחלקה מייצאת בנאי ליצירת עצם כאשר התוכן (שלם) והאבר הבא הם פרמטרים.
- המחלקה מייצאת שאילתות עבור התוכן והאבר הבא, ופקודות לשינוי האבר הבא, ולהדפסת תוכן הרשימה מהאבר הנוכחי.
- השדות מוגדרים כפרטיים – מוסתרים מהלקוחות.

בנאי constructor - תזכורת

- בנאי נכתב בדומה לשרות.
- שם הבנאי כשם המחלקה, והוא יכול לקבל פרמטרים כמו שרות אחר.
- בשונה משרות, הגדרת הבנאי אינה כוללת טיפוס מוחזר.
- הבנאי נקרא באופן הבא:

```
new <ClassName> ( <parameters>)
```

- נוצר עצם, אחר כך מופעל גוף הבנאי על העצם, ובסיום מוחזרת התייחסות לעצם הזה.

class IntCell

```
public class IntCell {
    private int cont;
    private IntCell next;

    public IntCell(int cont, IntCell next) {
        this.cont = cont;
        this.next = next;
    }

    public int cont() {
        return cont;
    }
```


class IntCell

```
    public IntCell next() {
        return next;
    }

    public void setNext(IntCell next) {
        this.next = next;
    }

    public void printList() {
        System.out.print("List: ");
        for (IntCell  y = this ;  y != null ; y = y.next())
                System.out.print(y.cont() + " ");
        System.out.println();
    }
}
```

דיון: "מקומה הטבעי" של `printList()`

- האם המחלקה `IntCell` מייצגת תא או רשימה?
- איך נממש את `printList()` כך ש"כל תא ידאג לעצמו" ?
- מה תכיל המחלקה `List` ?
- שימוש ב `toString`
- אז למה בכל זאת `printList()` מופיעה במחלקה `IntCell` ?

מחלקה לביצוע בדיקות

- כדי לבדוק שהמחלקה שכתבנו פועלת כנדרש, נכתוב מחלקה התחלתית לבדיקה, שתכיל שרות הראשי `main`.
- בהמשך הקורס נעסוק בנושא בדיקות (testing) אך כרגע נציין שעלינו לבחור מקרי בדיקה שמכסים אפשרויות שונות כדי שנוכל לגלות שגיאות (אם יש)
- חשוב! שגיאות של מחלקה או שרות מוגדרות בהקשר של *המוזה* של המחלקה. אם למחלקה (או לשרות שלה) אין חוזה מפורש לא ברור מהי ההתנהגות ה"נכונה" במקרי קצה

מחלקה לביצוע בדיקות

```
public class Test {
        public static void main(String[] args) {
                IntCell x = null;
                IntCell y = new IntCell(5,x);
                y.printList();
                IntCell z = new IntCell(3,y);
                z.printList();
                z.setNext(new IntCell(2,y));
                z.printList();
                y.printList();
        }
}
```


מחלקה לביצוע בדיקות – הפלט

```
List: 5
List: 3 5
List: 3 2 5
List: 5
```


מחלקות ושרותים מוכללים (גנריים)

- החל מגירסא 1.5 (נקראת גם 5.0) ג'אווה מאפשרת הגדרת מחלקות גנריות ושרותים גנריים.
- מחלקה גנרית מגדירה טיפוס גנרי, שמציין אחד או יותר משתני טיפוס (type variables) בתוך סוגריים משולשים.
- עקב ההוספה המאוחרת לשפה (והדרישה שקוד שנכתב קודם יוכל לעבוד ביחד עם קוד חדש), ומשיקולים של יעילות המימוש, כללי השפה לגבי טיפוסים גנריים הם מורכבים.

מחלקות ושרותים מוכללים (גנריים)

- כרגע נציג רק את המקרה הפשוט. בהמשך נחזור לדון בנושא ביתר פירוט.
- דוגמא ראשונה – הכללה של המחלקה `IntCell` לייצוג תא שתוכנו מטיפוס פרמטרי `T`, כך שכל התאים ברשימה הם מאותו הטיפוס.

Cell <T>

```
public class Cell <T> {
    private T cont;
    private Cell <T> next;

    public Cell(T cont, Cell <T> next) {
        this.cont = cont;
        this.next = next;
    }
```


Cell <T>

```
public T cont() {
    return cont;
}

public Cell <T> next() {
    return next;
}

public void setNext(Cell <T> next) {
    this.next = next;
}
```


Cell <T>

```
    public void printList() {
        System.out.print("List: ");
        for (Cell <T> y = this; y != null; y = y.next())
                System.out.print(y.cont() + " ");
        System.out.println();
    }
}
```


מה השתנה במחלקה?

- לכותרת המחלקה נוסף משתנה הטיפוס T (מקובל ששמות משתני טיפוס הם אות גדולה אחת).
- הטיפוס שמוגדר הוא `Cell <T>`
- הטיפוס של כל שדה, פרמטר, משתנה זמני, וכל טיפוס מוחזר של שרות שהיה `int` יוחלף ב T
- הטיפוס של כל שדה, פרמטר, משתנה זמני, וכל טיפוס מוחזר של שרות שהיה `Cell` יוחלף ב `Cell<T>`

שימוש בטיפוס גנרי

- כדי להשתמש בטיפוס גנרי יש לספק, בהצהרה על משתנה, ובקריאה לבנאי, טיפוס קונקרטי עבור כל משתנה טיפוס שלו.
  - לדוגמא: `Cell <Integer>`
- באנלוגיה להגדרת שרות (או שיגרה) וקריאה לה, משתנה טיפוס בהגדרת המחלקה מהווה מעין פרמטר פורמלי, והטיפוס הקונקרטי הוא מעין פרמטר אקטואלי.

שימוש בטיפוס גנרי

- הטיפוס הקונקרטי חייב להיות של עצם, כלומר אינו יכול להיות פרימיטיבי.
- אם רוצים ליצור למשל תאים שתוכנם הוא מספר שלם, לא ניתן לכתוב `Cell <int>`
- לצורך זה נזדקק לטיפוסים עוטפים (wrapper type)

טיפוסים עוטפים (wrappers)

- לכל טיפוס פרימיטיבי קיים בג'אווה טיפוס עצם:
  - ל float העוטף Float , ל double העוטף Double וכו'
- כל הטיפוסים העוטפים מקובעים (ערך של עצם לא משתנה).
- הטיפוסים העוטפים שימושיים כאשר יש צורך בעצם (למשל ביצירת אוספים של ערכים, ובשימוש בטיפוס גנרי).
- ג'אווה 1.5 מאפשרת מעבר אוטומטי בין טיפוס פרימיטיבי לטיפוס העוטף שלו – יצירת "קופסא" או הוצאה מהקופסה

```
Integer i = 0; // autoboxing
int n = i;     // autounboxing
```


בחזרה לשימוש בטיפוס גנרי

- נראה מחלקה שמשתמשת ב `Cell <T>` , שהיא הכללה של המחלקה שהשתמשה ב `IntCell` :

```
public class TestGen {
    public static void main(String[] args) {
            Cell <Integer> x = null;
            Cell <Integer> y =  new Cell<Integer>(5,x);
            y.printList();
            Cell<Integer> z= new Cell <Integer>(3,y);
            z.printList();
            z.setNext(new Cell <Integer>(2,y));
            z.printList();
            y.printList();
    }
}
```


עוד על שימוש בטיפוס גנרי

- ניתן להגדיר משתנה (שדה, משתנה זמני, פרמטר) גם מהטיפוס `Cell <Cell <Integer>>`
- מה מייצג הטיפוס הזה?
- דוגמא של הצהרה עם אתחול:

```
Cell <Cell <Integer> > q =
   new Cell <Cell <Integer>>
      (new Cell<Integer> (8,null), null);
```