

שימוש במחלקות קיימות

מטרת התרגיל:

- שימוש בחלקות קיימות הקיימות בג'אווה
- לדעת להפעיל פעולות על סמך ממשק המחלקה
- לדעת לזהות ולקרוא מימשק מחלקה על פי javaAPI

- (1 שימוש ב `awt.Point`)
- (2 שימוש ב `awt.Rectangle`)
- (2 שימוש ב `String`)
- (3 שימוש ב `JPanel`, `JFrame`)

בג'אווה קיימות חבילות תכנה וביניהם חבילת התכנה **Abstract Windowing Toolkit - AWT** כדי להשתמש במחלקה מתוך חבילת התכנה `awt` יש צורך לייבא אותה בעזרת `import`

מה זה API?

API הן ראשי תיבות של **Application Programming Interface**,

תרגיל 1: שימוש במחלקה `Point`

מטרת התרגיל: ליצור עצמים מטיפוס המחלקה `Point` ולבצע עליהם פעולות שקיימות בממשק המחלקה.

Point מייצגת נקודה במישור שתכונות שלה הן שיעורי `x,y` מטיפוס `int`.

לפניך ממשק חלקי של המחלקה:

| | |
|--|---|
| <code>Point(int x, int y)</code> | פעולה בונה נקודה במיקום <code>(x,y)</code> |
| <code>void move(int x, int y)</code> | הפעולה מעבירה את הנקודה להיות במקום החדש <code>(x,y)</code> על פי הערכים המעברים בפרמטר |
| <code>void translate(int dx,int dy)</code> | הפעולה מזיזה את הנקודה <code>dx</code> יחידות בכיוון ציר <code>x</code> ו <code>dy</code> יחידות בכיוון ציר <code>y</code> כך שמיקומה החדש הוא <code>(x+dx,y+dy)</code> |
| <code>public boolean equals(Point otherPoint)</code> | הפעולה מחזירה 'אמת' אם הנקודה זהה בערכים שלה לנקודה המעברת כפרמטר, 'שקר'-אחרת |
| <code>String toString()</code> | הפעולה מחזירה מחרוזת המתארת את הנקודה |

שלבי עבודה:

(1 צור מחלקה בשם `UsePoint`)

(2 ייבא את המחלקה `Point` לפני כותרת המחלקה : `import java.awt.Point;`)

(3 בפעולה הראשית הקלידו את שורות הקוד הבאות:

```
Point point1 = new Point(5,2);
System.out.println("point1 : " + point1);
```

(4 הרץ ובדוק מה מתקבל.

(5 כתוב הוראה ליצירת נקודה חדשה בשם `point2` במיקום `(2,7)` והדפיסו אותה.

(6) הוסף הוראה לבדיקה איזה נקודה יותר גבוהה, כלומר: לאיזה נקודה שיעור ה y שלה יותר גדול יותר. שים לב שיכול להיות מצב ששתי הנקודות באותו הגובה. הצג כפלט תוצאת בדיקה זו.

(7) הוסף הוראות להזזת הנקודה הנמוכה יותר עד שתגיע לגובה הנקודה הגבוהה יותר. כל פעם הזז את הנקודה יחידה אחת בכיוון ציר y. הדפס את הנקודה הנמוכה שזו שזזה בכל שלב של תזוזה.

```
point2 is higher
point1:java.awt.Point[x=5,y=3]
point1:java.awt.Point[x=5,y=4]
point1:java.awt.Point[x=5,y=5]
point1:java.awt.Point[x=5,y=6]
point1:java.awt.Point[x=5,y=7]
```

דוגמה לפלט:

(8) כתוב הוראה לבדיקה אם שתי הנקודות זהות בערך והצג כפלט את תוצאת הבדיקה.

(9) כתוב הוראה או הוראות להזזת את נקודה point2 למיקום הנקודה point1.

(10) הדפס את ערכי הנקודות אחרי ההזזה.

(11) כתוב את ההוראות הבאות:

```
boolean same = point1==point2;
boolean equal = point1.equals(point2);
System.out.println("is point1==point2 ? "+same);
System.out.println("is point1.equals(point2) ? "+equal);
```

(12) הרץ ובדוק. מה המסקנה?

כדי להשוות בין 2 עצמים יש להשתמש בפעולת equals
השוואה בעזרת פעולת == הינה בדיקה האם ההפניות הן לאותו העצם.

תרגיל 2: שימוש במחלקה Rectangle

מטרת התרגיל: ליצור עצמים מטיפוס המחלקה Rectangle ולבצע עליהם פעולות שקיימות בממשק המחלקה.

Rectangle מייצגת מלבן במישור שהצלעות שלנו מקבילות לצירים. התכונות שלו הן שיעורי (x,y) של נקודה שמאלית עליונה, רוחב - width וגובה - height. כל הגדלים הן מטיפוס שלם.

לפניך ממשק חלקי של המחלקה:

| | |
|--|---|
| Rectangle (int x, int y, int width, int height) | פעולה בונה מלבן במיקום (x,y) בעל רוחב width ואורך height. |
| boolean contains (Point p) | הפעולה מקבלת נקודה p ומחזירה 'אמת' אם המלבן מכיל את הנקודה, 'שקר' אחרת. |
| Rectangle intersection (Rectangle r) | הפעולה מקבלת מלבן r ומחזירה מלבן שהוא החיתוך בין המלבן למלבן r |
| Rectangle union (Rectangle r) | הפעולה מקבלת מלבן r ומחזירה מלבן שהוא האיחוד בין המלבן למלבן r |
| String toString () | הפעולה מחזירה מחרוזת המתארת את המלבן |

שלי עבודה:

- (1) צור מחלקה בשם **UseRectangle**.
- (2) ייבא את המחלקה java.awt.Rectangle , java.awt.Point .
- (3) בפעולה הראשית:
 - צור מלבן על פי ערכים שתבחר ונקודה על פי ערכים שתבחר.
 - הוסף משתנה בוליאני אשר יכיל פסוק בדיקה אם המלבן מכיל את הנקודה.
 - הדפס את נתוני המלבן, הנקודה והודעה עם המלבן מכיל את הנקודה או לא.
- (4) בפעולה הראשית הוסף את ההוראות הבאות:
 - צור עוד מלבן
 - צור מלבן שהוא החיתוך של המלבן הראשון והמלבן השני
 - צור מלבן שהוא האיחוד של המלבן הראשון והמלבן השני
 - הדפס את נתוני 4 המלבנים אחד אחרי השני.

```
run:
rect1:java.awt.Rectangle[x=20,y=50,width=40,height=30]
point1:java.awt.Point[x=40,y=60]
rect includes point
rect1:java.awt.Rectangle[x=20,y=50,width=40,height=30]
rect2:java.awt.Rectangle[x=40,y=30,width=60,height=40]
rectI:java.awt.Rectangle[x=40,y=50,width=20,height=20]
rectI:java.awt.Rectangle[x=20,y=30,width=80,height=50]
```

דוגמה לפלט:

** כדי ללמוד עוד על המחלקות :

- <http://docs.oracle.com/javase/7/docs/api/java/awt/Point.html>
- <http://docs.oracle.com/javase/7/docs/api/java/awt/Rectangle.html>

תרגיל 3: שימוש במחלקה String

המחלקה **String** מגדירה טיפוס שמכיל רצף של תווים.

דוגמאות לרצף תווים: "0123", "AA2", "abc" וכדומה.

למעשה כדי ליצור הפנייה לעצם מטיפוס **String** יש לכתוב: `String str1 = new String("abc");`

אולם בגלל שהשימוש במחלקה זו הוא כה שכיח ג'אוה מאפשרת ליצור הפנייה לעצם מטיפוס מחרוזת גם ללא המילה השמורה `new`, אלא כך: `String str2 = "abc"`.

לפניך ממשק חלקי של המחלקה String

| | | |
|---|---|---|
| | <code>s1.length();</code> | החזרת אורך המחרוזת |
| | <code>s1.charAt(k);</code> | החזרת תו במקום k במחרוזת |
| s1 בודקת אם s2 זהה לה | <code>s1.equals(s2);</code> | מחזיר 'אמת' אם תוכן המחרוזות שווה, 'שקר' - אחרת |
| s1 לפני s2 : אם שילי : אם שוות אפס : אם חיובי : אם s1 אחרי s2 | <code>s1.compareTo(s2);</code> | השוואה מילונית של 2 מחרוזות |
| ch – תו str – מחרוזת מיקום ראשון = 0 | <code>s1.indexOf(ch)</code> <code>s1.indexOf(str)</code> | החזרת המיקום הראשון של תו/תחילת תת-מחרוזת מתחילת המחרוזת. (-1) אם לא קיים. |
| ch – תו str – מחרוזת מיקום ראשון = 0 | <code>s1.indexOf(ch, k)</code> <code>s1.indexOf(str, k)</code> | החזרת המיקום הראשון של תו/תחילת תת-מחרוזת ממיקום k במחרוזת. (-1) אם לא קיים. |
| subStr – מחרוזת | <code>s1.startsWith(subStr)</code> | מחזירה 'אמת' אם תת-מחרוזת נמצאת בתחילת המחרוזת, 'שקר' – אחרת |
| subStr – מחרוזת | <code>s1.endsWith(subStr)</code> | מחזירה 'אמת' אם תת-מחרוזת נמצאת בסוף המחרוזת, 'שקר' – אחרת |
| ch – תו str – מחרוזת מיקום ראשון = 0 | <code>s1.lastIndexOf(ch)</code> <code>s1.lastIndexOf(str)</code> | החזרת המיקום האחרון של תו/תחילת תת-מחרוזת מסוף המחרוזת. (-1) אם לא קיים. |
| ch – תו str – מחרוזת מיקום ראשון = 0 | <code>s1.lastIndexOf(ch, k)</code> <code>s1.lastIndexOf(str, k)</code> | החזרת המיקום האחרון של תו/תחילת תת-מחרוזת ממיקום k מסוף המחרוזת במחרוזת. (-1) אם לא קיים. |
| | <code>s1.substring(k)</code> | מחזירה תת-מחרוזת ממיקום k ועד סוף המחרוזת. |
| p כולל מיקום k ולא כולל p | <code>s1.substring(k, p)</code> | מחזירה תת-מחרוזת ממיקום k ועד p |
| | <code>s1.replace(str1, str2)</code> | מחזירה מחרוזת חדשה בה כל מחרוזות str1 מוחלפות במחרוזות str2 |
| | <code>s1.replaceFirst(str1, str2)</code> | מחזירה מחרוזת חדשה בה הופעת str1 הראשונה מוחלפת במחרוזות str2 |
| | <code>s1.substring(k)</code> | מחזירה תת-מחרוזת מתחילתה ועד k |
| | <code>s1.substring(k,p)</code> | מחזירה תת-מחרוזת מ-k ועד p (ללא) |
| | <code>s1 = s1.toLowerCase();</code> | מחליף את כל אותיות המחרוזת לאותיות קטנות |
| | <code>s1 = s1.toUpperCase();</code> | מחליף את כל אותיות המחרוזת לאותיות גדולות |

שלבי עבודה:(1) צור מחלקה **UseString**

(2) כתוב את ההוראות הבאות בפעולה הראשית:

```
String s1 = "abc";
String s2 = "abc";
String s3 = new String("abc");
System.out.println("s1==s2: "+(s1==s2) );
System.out.println("s1.equals(s2): "+s1.equals(s2));
System.out.println("s1==s3: "+(s1==s3) );
System.out.println("s1.equals(s3): "+s1.equals(s3));
```

(3) הרץ ובודק מה הפלט.

מסקנה:להשוואת תוכן מחרוזות משתמשים בפעולה **equals** ולא בהוראת השוואה **==**.פתור את התרגילים הבאים בשימוש פעולות על מחרוזת:הנח שקיים קבוע: `final int N=6;`**תרגיל 1: שימוש בפעולה length**

כתוב פעולה אשר הקלט שלה N מחרוזות

עבור כל מחרוזת תציין את אורכה.

בנוסף תמצא את אורך המחרוזת הארוכה ביותר והקצרה ביותר.

תרגיל 2: שימוש בפעולה equalsכתוב פעולה אשר קולטת מחרוזת **first**.

לאחריה תקלט מספר שלם n ולאחריה עוד n מחרוזות.

התכנית תמנה ותודיע עמה מחרוזות מבין n המחרוזות זהות למחרוזת **first**.**תרגיל 3: שימוש בפעולה compareTo**כתוב פעולה אשר קולטת מחרוזת **first**.

לאחריה תקלט מספר שלם n ולאחריה עוד n מחרוזות.

עבור כל מחרוזת מ n המחרוזות הפעולה תודיע אם המחרוזת קודמת למחרוזת **first**, זהה לה או באה אחריה בסדר מילוני.

תרגיל 4: שימוש ב indexOf

כתוב פעולה הקולטת שתי מחרוזות str1, str2 .
הפעולה תבדוק ותודיע האם המחרוזת str2 מופיעה במחרוזת str1 .

תרגיל 5: שימוש ב indexOf

כתוב פעולה הקולטת שתי מחרוזות str1, str2 .
הפעולה תבדוק ותודיע כמה פעמים המחרוזת str2 מופיעה במחרוזת str1 .

תרגיל 6: בדיקת @ בכתובת דוא"ל

כתובת דוא"ל מכילה את התו @ פעם אחת בלבד. לא במקום ראשון ולא באחרון.
כתוב פעולה הקולטת מחרוזת שמהווה כתובת דוא"ל.
הפעולה תבדוק האם התו '@' מופיע במחרוזת או לא.
במידה ומופיע תודיע האם הוא קיים רק פעם אחת ולא במקום ראשון ולא במקום אחרון.

תרגיל 7: בדיקת ת.ז.

כתוב פעולה אשר קולטת מחרוזת שמהווה מספר של תעודת זהות.
תעודת זהות חייבת להכיל רק ספרות ואורכה הוא 9.
הפעולה תבדוק ותודיע אם המחרוזת יכולה להיות מספר של תעודת זהות או לא.

תרגיל 8: בדיקת מספר טלפון

כתוב פעולה אשר קולטת מחרוזת שמהווה מספר טלפון ברשת סלולרית.
הפעולה תבדוק שהמחרוזת מתחילה בתווים "05". במקום הרביעי מופיע '-' ואחרי סימן זה יש עוד 7 מספרים.
הפעולה תציג כפלט הודעה אם המחרוזת מהווה מספר הטלפון ברשת סלולרית או לא.