מערכים ומחרוזות תווים (Arrays & Strings)

מערכים – Arrays

מחרוזות תווים – Strings

© 2007 Zindell Technologies, Ltd.

מערכים - Arrays

הקדמה

מערך ב-JAVA נחשב לאובייקט. אובייקט שמייצג אוסף מסודר של אובייקטים (או ערכים) מאותו טיפוס, שמסודרים בשורה, ושהגישה לכל אחד מהם אפשרית באמצעות מספר ה-index המתאים. מספר האינדקס שניתן לאיבר הראשון שווה ל-0, ומספר האינדקס שניתן לאיבר האחרון קטן באחד ממספר האיברים במערך (כמו בשפת התיכנות C).

כדי לייצור מערך יש, תחילה, להצהיר על משתנה חדש שיוכל להכיל בתוכו את ה-reference למערך שיוצרים (תזכורת: מערך ב-JAVA נחשב לאובייקט, ולכן יש לו reference).

כדי לייצור משתנה שיוכל להכיל בתוכו reference לאובייקט, מצהירים על המשתנה בדומה להצהרה על משתנה מטיפוס בסיסי. לדוגמא, בשורה הבאה מצהירים משתנה בשם tav אשר יוכל להכיל reference לאובייקט מטיפוס Car:

```
Car  tav;
```

כדי לייצור משתנה שיוכל להכיל בתוכו reference למערך יש להוסיף לפני שם המשתנה או אחריו צמד של סוגריים מרובעות. לדוגמא, בשורה הבאה נוצר משתנה בשם vec אשר יוכל להכיל בתוכו reference למערך:

```
Car  []vec;
```

את צמד הסוגריים המרובעות ניתן למקם או לפני השם של המשתנה או אחריו. ולכן, ניתן היה לרשום את השורה הבאה גם כך:

```
Car vec[];
```

בשלב זה עדיין לא הוקצה מקום בזיכרון עבור המערך. כל מה שנעשה בשלב זה הוא יצירת משתנה שיוכל להכיל בתוכו reference למערך שנייצור בהמשך.

אוסף הערכים שמוחזק במערך יכול להיות אוסף של primitive type values (לדוגמא: מערך של ערכים מטיפוס int) או אוסף של references לאובייקטים (לדוגמא: מערך של references לאובייקטים שנוצרו מהמחלקה Student).

© 2007 Zindell Technologies, Ltd.

קיים שוני באופן יצירתו של מערך בכל אחד משני המקרים.

מערך של references לאובייקטים

המערך המכיל אוסף של references לאובייקטים נחשב לאובייקט אשר יש לו reference שניתן לאיחסון במשתנה מתאים. כדי לייצור משתנה שיוכל להכיל בתוכו reference למערך כגון זה יש לרשום את שם המחלקה ואחריה את שם המשתנה בצירוף צמד סוגריים מרובעות. לדוגמא:

```
Car vec[];
```

בשלב זה,

מה שנוצר הוא רק משתנה. מקום בזיכרון ששמו vec, אשר יוכל בהמשך לאחסן בתוכו reference למקום אחר שבו יהיה אובייקט שבחדריו references לאובייקטים. vec כמשתנה יאחסן, למעשה, reference למערך.

כעת יש להקצות את המקום בזיכרון שבו יאוחסן המערך(המערך נחשב לאובייקט) אשר מהווה אוסף של מקומות בזיכרון אשר מכילים (כל אחד מהם) references לאובייקטים. הקצאת המקום בזיכרון שישמש לאיחסון ה-references לאובייקטים נעשית באופן הבא: יש לרשום new ואחריו את שם המחלקה ומייד בצמוד אליה (אחריה) צמד סוגריים מרובעות כשבתוכן מספר שלם שמשמעות תהיה מספר המקומות בזיכרון שיוקצו לאיחסון ה-references, כלומר, מספר ה-references שניתן יהיה לאחסן במערך. לדוגמא (המשך הדוגמא הקודמת):

```
vec = new Car[10];
```

בביצוע שורה זו יוצרים מערך שיש בו 10 מקומות בזיכרון שיכולים להכיל (כל אחד מהם) reference לאובייקט מטיפוס Car. בשלב זה, כל אחד מעשרת המקומות עדיין לא מכיל reference לאובייקט מטיפוס Car. בשלב זה, כל אחד מעשרת המקומות מכיל null. הפניה לכל אחד מעשרת המקומות נעשית בדומה ל-C: בציון שם המערך, vec בדוגמה זו, בתוספת סוגריים מרובעות כשבתוכן מספר האינדקס של המקום אשר אליו פונים. כך למשל כתיבת השורה vec[0] מהווה פנייה למקום הראשון במערך ודומה במעמד שיש לה בתכנית לכתיבת שמו של משתנה מטיפוס Car אשר מכיל reference לאובייקט. כתיבת שמו של משתנה מטיפוס Car דומה לכתיבת שמו של המערך בצירוף מספר אינדקס בתוך

© 2007 Zindell Technologies, Ltd.

סוגריים מרובעות. לשתי האפשרויות יש ערך שהוא reference מטיפוס Car. כעת, כדי שכל אחד מעשרת המקומות יכיל reference לאובייקט מטיפוס Car יש לייצור 10 אובייקטים מטיפוס Car ולאחסן את ה-reference של כל אחד מהם בכל אחד מעשרת המקומות. יצירתו של כל אחד מעשרת האובייקטים תיעשה באמצעות הפעלת אחד ה-constructors שמוגדרים במחלקה Car. לדוגמא (המשך הדוגמא הקודמת):

```
for (int index=0; index<=9; index++)
{
    vec[index]=new Car();
}
```

באמצעות לולאה זו יוצרים (בדוגמא האמורה) 10 אובייקטים מטיפוס Car, ומאחסנים את ה-reference של כל אחד מהם בכל אחד ממקומות המערך. יצירת כל אחד מעשרת האובייקטים נעשית באמצעות המילה השמורה new ואחד ה-constructors.

כעת, לפני שנמשיך, נסכם את שלושת השלבים ביצירת מערך של אובייקטים:

שלב ראשון, הקצאת המקום בזיכרון שיהווה משתנה שיוכל להכיל בתוכו reference למערך, אשר מהווה אובייקט, לדוגמא:

```
Car vec[];
```

שלב שני, הקצאת המקום בזיכרון לאיחסון ה-references לאובייקטים של המערך, והכנסת ה-reference שלו אל תוך המשתנה שנוצר בשלב הקודם. המשך הדוגמא:

```
vec = new Car[10];
```

בשלב זה מוקצה הזיכרון לאובייקט שנחשב למערך.

את שני השלבים הראשונים ניתן היה לאחד לשלב אחד, לדוגמא:

© 2007 Zindell Technologies, Ltd.

```
Car vec = new Car[10];
```

שלב שלישי, יצירת כל אחד מהאובייקטים של המערך, ואיחסון ה-reference שלו במקום המתאים במערך. בשלב זה יש להיעזר, בדרך כלל, בלולאה אשר בכל איטרציה תיצור אובייקט חדש, ותאחסן את ה-reference שלו במקום המתאים במערך. המשך הדוגמא:

```
for (int index = 0; index <= 9; index++)
{
    vec[index] = new Car();
}
```

שים/שימי לב לכך שהשלב השלישי איננו חייב להתבצע באופן מיידי לאחר יצירת המערך (יצירת האובייקט שמשמש כמערך של references לאובייקטים). ניתן, בהחלט, למלא את תאיו של המערך ב-references לאובייקטים במהלך פעולתה של התכנית ובהתאם לצורך.

בדוגמה לעיל, מוצגת תכנית אשר מדגימה את השימוש במערך של references לאובייקטים:

```
class Kushkush
{
	private int x,y;

	public Kushkush()
	{
		this(0,0);
	}
```

© 2007 Zindell Technologies, Ltd.

```
public Kushkush(int a, int b)
{
        x=a;
        y=b;
}


public Kushkush(int a)
{
        this(a,a);
}


public void set(int a,int b)
{
        x=a;
        y=b;
}
public boolean equals(Kushkush otherKushkush)
{
        if (otherKushkush.x==x && otherKushkush.y==y)
```

© 2007 Zindell Technologies, Ltd.

```
                return true;
            else
                return false;
        }
        public void display()
        {
            System.out.println("\nx="+x+"   y="+y);
        }
}


class ClassVecDemo
{
    public static void main(String args[])
    {
        int index;
        Kushkush kushkushs[];
        // creating a place to put reference to memory location
        // in which references to the Kushkush objects can be placed
        kushkushs = new Kushkush[10];
```

© 2007 Zindell Technologies, Ltd.

```
    // creating 10 places that in each one of them
    // a refernce to a Kushkush object can be placed
    // the address (reference) of the place in which 10 places
    // locate is returned
    for (index=0; index<10; index++)
    {
      // assigning in each one of the 10 places
      // a reference to a new Kushkush object
      kushkushs[index] = new Kushkush(index+1,index+1);
    }
    for (index=0; index<10; index++)
    {
      // printing the information of each one of the
      // 10 Kushkush objects
      kushkushs[index].display();
    }
  }
}
```

© 2007 Zindell Technologies, Ltd.

מערך של ערכים מטיפוס בסיסי

יצירתו של מערך המכיל אוסף של ערכים מטיפוס בסיסי (לדוגמא: מערך של ערכים מספריים מטיפוס int), דומה ליצירתו של מערך המכיל אוסף של references לאובייקטים. הדמיון הוא בקיומם של שני השלבים הראשונים. השוני הוא בשלב השלישי.

ביצירת מערך של ערכים מטיפוס בסיסי (int, float וכדומה), ב-JAVA יש שני שלבים בלבד. בשלב ראשון מצהירים על משתנה אשר יוכל להכיל בתוכו reference למערך. מערך של ערכים מטיפוס בסיסי ב-JAVA הוא אוסף של מקומות בזיכרון שבכל אחד מהם מאחסנים ערך מטיפוס בסיסי, ויחדיו הם מהווים אובייקט.

דוגמא לביצוע השלב הראשון:

```
int numbers[];
```

בשלב זה נוצר משתנה ששמו numbers ושבתוכו ניתן יהיה לאחסן reference למערך מטיפוס int.

כעת, ניתן לעבור אל השלב השני, שבו יוצרים את המערך (זהו המקום בזיכרון שבו יאוחסנו ערכי המערך), ומאחסנים את ה-reference שלו במשתנה שנוצר בשלב הראשון. יצירת המערך נעשית באמצעות המילה new וכתיבת שם הטיפוס הבסיסי (אשר יהיה – למעשה - הטיפוס של כל אחד מחדרי המערך), כשמייד אחריו בתוך סוגריים מרובעות מציינים את גודלו.

דוגמא לביצוע השלב השני:

```
numbers = new int[100];
```

בשלב זה, נוצר מקום בזיכרון שמכיל 100 מקומות לאיחסון 100 ערכים מטיפוס int. ה-reference למקום שנוצר מאוחסן במשתנה numbers. יש לשים לב להבדל לעומת מערך של references לאובייקטים. במערך של ערכים מטיפוס בסיסי (כדוגמת הטיפוס int) כל אחד ממקומות המערך מכיל ערך מטיפוס בסיסי, ולא reference לאובייקט כפי שהיה במערך

© 2007 Zindell Technologies, Ltd.

של references לאובייקטים. מסיבה זו, כאשר יוצרים מערך של ערכים מטיפוס בסיסי לא קיים הצורך בשלב השלישי שהיה קיים ביצירתו של מערך של references לאובייקטים.

כמו כן, גם במערך של ערכים מטיפוס בסיסי ניתן לבצע את שני השלבים הראשון והשני ביחד.

לדוגמה:

```
int numbers[] = new int[100];
```

דוגמא ליצירת מערך של ערכים מטיפוס בסיסי:

```
int size = 25;

char tavs[] = new char[size];
```

בדוגמה זו, לאחר הפעלת האופרטור-new יווצרו 25 מקומות בזיכרון, ובכל אחד מהם ניתן יהיה לאחסן ערך מטיפוס char. יש לשים לב לכך שנוצרים פה 25 מקומות בזיכרון לאיחסון 25 ערכים ולא references, כי char הוא טיפוס בסיסי ולא מחלקתי.

בתכנית הבאה ניתן לראות הדגמה ליצירת מערך מטיפוס בסיסי ולאופן השימוש בו :

```
class PrimitiveTypesVecDemo
{
	public final static int SIZE = 10;

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

© 2007 Zindell Technologies, Ltd.

```
int numbers[];

// declaring an array variable

int index,sum=0;

numbers = new int[SIZE];

// initializing the array



// The upper two lines could be together:

// int numbers[] = new int[SIZE];



for (index=0; index<SIZE; index++)

{

        numbers[index] = index+1;

}





            for (index=0; index<SIZE; index++)

{

        sum += numbers[index];

}
```

© 2007 Zindell Technologies, Ltd.

```
            System.out.println(
                        "\nThe sum of all of the numbers is : " + sum);

      }
}
```

תכונה חשובה, שיש למערכים של ערכים מטיפוס בסיסי ב-JAVA (תכונה שמהווה הבדל נוסף לעומת מערכים של ערכים מטיפוס בסיסי ב-C) היא שכאשר יוצרים מערך של ערכים מטיפוס בסיסי ב-JAVA, אז כל אחד מהחדרים שבו מאותחל בערך ברירת המחדל עפ"י הטיפוס. כך לדוגמה, שורות הקוד הבאות יגרמו ליצירת מערך של 25 ערכים מטיפוס char, אשר כבר עם היווצרות המערך יהיו שווים ל-0.

```
int size = 25;
char tavs[] = new char[size];
```

אם יוצרים ב-JAVA מערך של references לאובייקטים אז כל אחד מהמקומות במערך אשר אמור להכיל reference לאובייקט יאותחל ב-null. כל עוד לא יווצר אובייקט כדי להכניס את ה-reference שלו למקום מסויים במערך, ימשיך אותו מקום להכיל null.

תכונה בולטת נוספת שיש למערכים ב-JAVA היא שלא ניתן לחרוג מגבולותיו. בעוד שב-C ניתן היה לחרוג מגבולותיו של המערך בטעות (תזכורת למי שמכיר את שפת התכנות C: ב-C מערך מהווה רצף של מקומות בזיכרון, ובתור שכזה, חריגות מגבולותיו בהחלט תיתכן) ב-JAVA, כיוון שמערך מהווה אובייקט, לא ניתן לחרוג מגבולותיו. ניסיון ב-JAVA לפנות לתא במערך באמצעות מספר אינדקס שגדול ממספר האינדקס של התא האחרון יגרום לשגיאה.

© 2007 Zindell Technologies, Ltd.

תכונה נוספת שמבדילה מערכים ב-JAVA ממערכים ב-C היא האפשרות להכניס לתוך שמו של מערך (זהו למעשה שם של משתנה אשר מכיל reference למערך) את הערך null. ב-C, כיוון ששמו של מערך נחשב ל-constant pointer, פעולה זו לא הייתה אפשרית לביצוע.

שילוב ההצהרה על המערך בשורה שיש בה הצהרה על משתנים

את שלב ההצהרה על המשתנה שישמש להחזקת ה-reference למערך (כלומר, את יצירת המשתנה שיכיל reference למערך, ויהווה בכך שם של מערך) ניתן לשלב בשורה אחת עם הצהרות על משתנים מאותו טיפוס.

לדוגמא,

```
int numbers[], num1, num2, incomes[];
```

בשורה זו נוצרו שני מערכים ושני משתנים מטיפוס int.

בשלב הראשון של יצירת המערך, שלב ההצהרה על המשתנה שיכיל את ה-reference אליו, ניתן למקם את הסוגריים המרובעות גם אחרי השם שניתן למערך(שם המשתנה), וגם לפניו.

לדוגמא:

```
int []vec;
```

או,

```
int vec[];
```

שתי האפשרויות מקובלות, ובשתיהן נוצר משתנה חדש ששמו vec, אשר יכול לאחסן בתוכו reference למערך (תזכורת: מערך ב-JAVA נחשב לאובייקט). עם זאת, קיים בכל זאת, הבדל בין שתי האפשרויות. אם שורת ההצהרה על המערך כוללת גם הצהרות של משתנים אז למיקום הסוגריים המרובעות יש משמעות.

אם הסוגריים ממוקמות אחרי השם שניתן למערך, אז כל השמות שיבואו אחרי שם המערך ואחרי הסוגריים

© 2007 Zindell Technologies, Ltd.

המרובעות שלו ייחשבו לשמות של משתנים פשוטים שהטיפוס שלהם זהה לטיפוס של המערך. לדוגמא:

```
int numbers[], num1, num2;
```

בדוגמא זו נוצרים שני משתנים מטיפוס int, ומשתנה נוסף אשר יוכל להכיל reference למערך מטיפוס int שניתן יהיה לייצור בהמשך.

אם הסוגריים ממוקמות לפני השם שניתן למשתנה שמתעתד להכיל reference למערך אז כל השמות שיופיעו אחריו (תזכורת: מערך ב-JAVA נחשב לאובייקט) יהוו גם הם משתנים שיכולים להכיל – כל אחד מהם – reference למערך. לדוגמא:

```
int []numbers, num1, num2;
```

בדוגמא זו, שלושת השמות (numbers -ו num1, num2) יהוו כל אחד מהם משתנה אשר אמור להכיל בתוכו reference למערך של ערכים מטיפוס int. במקרה זה, num1 ו- num2 לא יהוו משתנים פשוטים מטיפוס int. הם יהוו, בדומה ל-numbers, משתנים אשר כל אחד מהם יכול להכיל בתוכו reference למערך.

איתחול מערך

איתחולו של מערך יכול להיעשות בדומה ל-C באופן הבא:

```
int [] incomes = {1000, 2000,14000, 30000, 60000};
```

בכתיבת שורה זו (למשל) יווצר באופן אוטומטי מערך מטיפוס int, בגודל 5, אשר יכיל את 5 הערכים שמופיעים בתוך הסוגריים המסולסלות.

הבדל בולט ב-JAVA לעומת C הוא שניתן לאתחל את המערך גם בערכים אשר יהיו ידועים רק בזמן הריצה. ניתן, למשל, לרשום במקום כל אחד מהערכים המספריים ביטויים שיהיה להם ערך רק בזמן הריצה. לדוגמא:

```
int []incomes = {1000, num1+num2, 5500};
```

בשורה זו, למשל, יווצר מערך מטיפוס int, בגודל 3, אשר יכיל את שלושת הערכים שכתובים בתוך הסוגריים המסולסלות,

© 2007 Zindell Technologies, Ltd.

וזאת כאשר ערכם המדוייק יהיה ידוע רק בזמן ריצה.

במערך של references לאובייקטים, כל איבר (חדר) במערך מתנהג בדיוק כמו class type variable. כלומר, כל איבר במערך פועל כמו משתנה מטיפוס מחלקתי. לאחר הקצאת המקומות אשר יוכלו להכיל references לאובייקטים, כל אחד מהם יאותחל ב-null. גם מערך של references לאובייקטים ניתן לאתחל. ניתן לבצע את איתחולו באמצעות ה-references שמוחזרים על ידי המילה השמורה new שתשמש ליצירת כל אחד מהאובייקטים של המערך. לדוגמא:

```
Car cars[] = {new Car(), new Car(2, 3, 4)};
```

בדוגמא זו, למשל, המשתנה cars מכיל כעת reference למערך (תזכורת: מערך נחשב לאובייקט) בגודל 2. המערך מכיל 2 מקומות לאיחסון 2 references לשני אובייקטים מטיפוס Car, אשר נוצרים באמצעות המילה השמורה new בזמן ריצת התכנית.

כיוון שכל אחד משני המקומות של המערך מכיל reference לאובייקט, ניתן גם לאתחל את המערך באמצעות משתנים שמכילים כבר references של אובייקטים.

לדוגמא:

```
Car car1 = new Car(2,3,4);

Car car2 = new Car(4,3,1);

Car cars[] = {car1, car2};
```

בדוגמא זו נוצר מערך של references מטיפוס Car, אשר מכיל references לאובייקטים מטיפוס Car אשר ה-references שלהם כבר מאוחסנים בשני משתנים מטיפוס המחלקה Car.

בתכנית הבאה ניתן לראות הדגמה לפעולת האיתחול שניתן לבצע למערך מטיפוס מחלקתי.

```
class Punc

{
```

© 2007 Zindell Technologies, Ltd.

```
private int x,y;


public Punc()
{
	this(0,0);
}


public Punc(int a, int b)
{
	x=a;
	y=b;
}


public Punc(int a)
{
	this(a,a);
}


public void set(int a,int b)
{
```

© 2007 Zindell Technologies, Ltd.

```
        x=a;
        y=b;
    }
    public boolean equals(Punc otherPunc)
    {
        if (otherPunc.x==x && otherPunc.y==y)
            return true;
        else
            return false;
    }
    public void display()
    {
        System.out.println("\nx="+x+"   y="+y);
    }
}


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

© 2007 Zindell Technologies, Ltd.

```
{
    int index;

    Punc puncs[];
    // creating a place to put reference to
    // memory location in which the references to
    // the Punc objects can be placed

    puncs = new Punc[10];
    // creating 10 places that in each one of them
    // a refernce to a Punc object can be placed
    // the address (reference) of the place in which
    // 10 places locate is returned

    for (index=0; index<10; index++)
    {
        // assigning in each one of the 10 places
        // a reference to a new Punc object
        puncs[index] = new Punc(index+1,index+1);
    }
```

© 2007 Zindell Technologies, Ltd.

```
Punc superPunc = new Punc(7,7);

Punc []otherPuncs =  {Puncs[0],Puncs[1], new Punc(20,20),superPunc};

for (index=0; index<4; index++)

{

        // printing the information of each one of

                              // the 4 Punc objects

        otherPuncs[index].display();

}


// notice that changing the object superPunc will be

// seen in printing the array otherPuncss since the

// fourth element in that array is the address of an

// object which its address is also in the superPunc

// variable

superPunc.set(72,72);

System.out.print("\nThe Punc objects that their addresses ");

System.out.println("stored in the array after changing");

for (index=0; index<4; index++)

{
```

© 2007 Zindell Technologies, Ltd.

```
                // printing the information of each one of
                         // the 10 Punc objects
                otherPuncs[index].display();
            }
        }
}
```

השימוש במערכים

כל אחד מתאי מערך ב-JAVA מתפקד כמשתנה לכל דבר מטיפוס אותו מערך, בין אם הטיפוס מחלקתי, ובין אם הטיפוס בסיסי.

לדוגמא:

```
int []incomes = new int[100];
incomes[2] = 2500;
incomes[3] = incomes[2] + 1000;
int value = incomes[3];
```

מספר האינדקס שעושים בו שימוש חייב להיות מספר שלם.

מערך ב-JAVA הוא אובייקט. לאחר יצירתו מאותחלים איבריו. אם המערך מכיל אוסף של references לאובייקטים אז איבריו מאותחלים ב-null, ואם המערך מכיל אוסף של ערכים מטיפוס בסיסי אז איבריו מאותחלים בערך ברירת המחדל של אותו טיפוס בסיסי (לדוגמא: מערך שכולל 10 ערכים מטיפוס boolean יאותחל בעת יצירתו ויכיל 10 ערכי false, כיוון ש-false הוא ערך ברירת המחדל של הטיפוס boolean).

© 2007 Zindell Technologies, Ltd.

טווח האינדקסים של כל מערך מתחיל ב-0, וממשיך עד למספר שקטן ב-1 מגודלו של המערך. כך לדוגמא, אם במערך יש length איברים, אז מספר האינדקס של האיבר הראשון שווה ל-0, ומספר האינדקס של האיבר האחרון שווה ל-length-1. כל חריגה מטווח האינדקסים של מערך נתון תגרום לשגיאה.

כיוון שכל מערך מהווה גם אובייקט, יצרו מתכנני של השפה משתנה בשם length, אשר קיים בכל מערך, ומכיל את מספר האיברים שיש בו. הגישה למשתנה זה נעשית באופן שדומה לאופן שבו ניגשים למשתנה שנמצא בתוך אובייקט. יש לרשום ביטוי שערכו הוא ה-reference למערך (לדוגמא: שם של משתנה שמכיל בתוכו את ה-reference למערך) בצירוף נקודה מפרידה, ומייד אחריה את שמו של המשתנה שנמצא בתוך אובייקט המערך ושאליו מנסים לגשת (לדוגמא: length).

דוגמא:

```
int []incomes = new int[100];

for (int index=0; index<incomes.length; index++)

{

     incomes[index] = index * 1000;

}
```

המשתנה length קיים בתוך כל אובייקט שהוא מערך, והוא read only. כלומר, לא ניתן לשנות את ערכו.

התכנית הבאה מדגימה את השימוש במשתנה length.

```
class Choonk

{

      private int x,y;
```

© 2007 Zindell Technologies, Ltd.

```
public Choonk()
{
        this(0,0);
}

public Choonk(int a, int b)
{
        x=a;
        y=b;
}

public Choonk(int a)
{
        this(a,a);
}

public void set(int a,int b)
{
        x=a;
        y=b;
```

© 2007 Zindell Technologies, Ltd.

```
	}
	public boolean equals(Choonk otherChoonk)
	{
		if (otherChoonk.x==x && otherChoonk.y==y)
			return true;
		else
			return false;
	}
	public void display()
	{
		System.out.println("\nx="+x+"   y="+y);
	}
}

class LengthDemo
{
	public static void main(String args[])
	{
		int index;
```

© 2007 Zindell Technologies, Ltd.

```
Choonk choonks[];

// creating a place to put reference to memory

// locations in which references to the Choonk

// objects can be placed


Choonks = new choonk[10];

// creating 10 places that in each one of them

// a refernce to a Choonk object can be placed

// the address (reference) of the place in which 10

// places locate is returned


for (index=0; index<choonks.length; index++)

{

	// length is an instance variable

	// it is a read only variable

	// assigning in each one of the 10 places

	// a reference to a new Choonk object

	choonks[index] = new Choonk(index+1,index+1);

}
```

© 2007 Zindell Technologies, Ltd.

```
        for (index=0; index<choonks.length; index++)
        {
            // printing the information of each one of
            // 10 Choonk objects
            choonks[index].display();
        }
    }
}
```

העתקת איברי מערך

כאשר מבצעים משפט השמה שבו ערכו של משתנה (שמכיל reference למערך) מושם אל תוך משתנה אחר (שגם הוא מסוגל להכיל reference למערך) מקבלים מצב שבו יש בשני משתנים שונים reference זהה לאותו מערך.

לדוגמה,

```
int vec1[] = {1,2,3};
int vec2[] = {8,7,6,5};
vec1 = vec2;
```

שתי השורות הראשונות יצרו שני מערכים מטיפוס int, וגרמו לאיחסון ה-reference של כל אחד מהם במשתנים vec1, ו-vec2. כתוצאה ממשפט ההשמה האחרון, גם vec1 וגם vec2 מכילים את אותו reference, לאותו מערך (המערך שנוצר בשורה השניה), והמערך שנוצר בשורה הראשונה, ואוחסן בתוך המשתנה vec1, נאסף על ידי ה-Garbage Collector, ומושמד.

© 2007 Zindell Technologies, Ltd.

הדרך להעתקת ערכים של מערך אחד למערך שני היא באמצעות הפעלת המתודה arraycopy אשר מוגדרת כמתודה סטטית במחלקה System:

```
public static void arraycopy(Object src,
                             int src_position,
                             Object dst,
                             int dst_position,
                             int length)
```

מתודה זו מעתיקה את הערכים שנמצאים במערך src אל המערך שה-reference שלו מאוחסן במשתנה dst. אל תוך הפרמטר length נשלח מספר האיברים המועתקים, וההעתקה מתחילה באיבר שמספר האינדקס שלו הוא src_position אשר מועתק אל המערך dst אל המקום שמספר האינדקס שלו הוא dst_position.

לדוגמא:

```
int []vec1 = {1,2,3,4,5};
int []vec2 = new int[vec1.length];
System.arraycopy(vec1, 0, vec2, 0, vec1.length);
```

השורה השלישית גורמת להעתקת כל ערכיו של המערך vec1 אל המערך vec2 כך ששניהם נהיים זהים. המתודה arraycopy מוגדרת כ-public כדי שניתן יהיה להשתמש בה מכל מחלקה. יש לשים לב לכך שמתודה זו יכולה לשמש לביצוע העתקה של כל מערך מכל טיפוס, בין אם הוא מטיפוס בסיסי, ובין אם הוא מטיפוס מחלקתי. במקרה האחרון, יש לשים לב לכך שמה שמועתק הם references לאובייקטים. אם בפעולתה של המתודה תתבצע חריגה מגבולותיו של אחד

© 2007 Zindell Technologies, Ltd.

המערכים אז תיווצר שגיאה.

מערך דו-מימדי

כדי לייצור ב-JAVA מערך דו-ממדי, יש לייצור מערך חד ממדי שכל אחד מאיבריו הוא מערך חד ממדי בפני עצמו. הדרך לעשות זאת מודגמת בתכנית הבאה:

```
class TwoDimArray
{
    public static void main(String args[])
    {
        int twoDim[][] = new int [10][];
            //this line creates one dimension array
        //with 10 empty places to hold 10
        //references of 10 other one dimension arrays

        int i,j;
        for (i=0;i<10;i++)
        {
            twoDim[i] = new int[10];
            //this line creates an array of 10 integer
            //values and puts its reference into the
```

© 2007 Zindell Technologies, Ltd.

```
            //appropriate place in the twoDim array
        }
        for (i=0;i<10;i++)
        {
            for (j=0;j<10;j++)
                twoDim[i][j]=(i+1)*(j+1);
        }
        for (i=0;i<10;i++)
        {
            for (j=0;j<10;j++)
            {
                System.out.print(twoDim[i][j]+" ");
            }
            System.out.println(" ");
        }
    }
}
```

את המערך הדו ממדי ניתן לייצור גם בדרך מקוצרת יותר, כפי שמציגות שתי התכניות הבאות.. התכנית הראשונה מציגה יצירתו של מערך דו מימדי מטיפוס בסיסי.

© 2007 Zindell Technologies, Ltd.

```
class TwoDimArrayShort
{
	public static void main(String args[])
	{
		int twoDim[][] = new int [10][10];
		int i,j;
		for (i=0;i<10;i++)
		{
			for (j=0;j<10;j++)
				twoDim[i][j]=(i+1)*(j+1);
		}
		for (i=0;i<10;i++)
		{
			for (j=0;j<10;j++)
			{
				System.out.print(twoDim[i][j]+" ");
			}
			System.out.println(" ");
		}
	}
```

© 2007 Zindell Technologies, Ltd.

```
}
```

התכנית הבאה מציגה את יצירתו של מערך דו מימדי אשר איבריו הם references לאובייקטים.

```
class Dogy
{
    public int size,age;
    public String display()
    {
        String result = "size=" + size + "age=" + age;
        return result;
    }
}


public class TwoDimArrayShort2
{
    public static void main(String args[])
    {
        Dogy twoDim[][] = new Dogy [3][3];
        //This is stage2 + stage1
        //This line creates a 3X3 array to hold 9 references
```

© 2007 Zindell Technologies, Ltd.

```
//of 9 objects

int i,j;

for (i=0;i<3;i++)
{
        for (j=0;j<3;j++)
        {
            twoDim[i][j] = new Dogy();
            twoDim[i][j].size=i;
            twoDim[i][j].age=j;
        }
}

for (i=0;i<3;i++)
{
        for (j=0;j<3;j++)
        {
            System.out.print(twoDim[i][j].display());
        }
```

© 2007 Zindell Technologies, Ltd.

```
                System.out.println(" ");
            }
        }
}
```

מחרוזות תווים – Strings

הקדמה

מחרוזת תווים ב-JAVA מאוחסנת בתוך אובייקט שיוצרים מהמחלקה

java.lang.String.

כדי לייצור אובייקט חדש ממחלקה זו כדי שייצג מחרוזת תווים חדשה ניתן להשתמש ב-new בצירוף קריאה להפעלת ה-constructor המתאים, בדומה ליצירת אובייקטים ממחלקות אחרות:

```
String str = new String("Israel");
```

דרך נוספת ליצירת אובייקט מטיפוס המחלקה String היא לרשום מחרוזת תווים. ב-JAVA, כאשר כותבים מחרוזת תווים, יוצרים בכך, למעשה, אובייקט חדש מטיפוס המחלקה String, וערכה של המחרוזת הנכתבת הנו reference לאובייקט שנוצר כדי לייצגה. מסיבה זו, את שורת הפקודה האחרונה ניתן היה גם לרשום באופן הבא:

```
String str = "Israel";
```

הפונקציות הבונות - Constructors

במחלקה String קיימים מספר constructors אשר מאפשרים לייצור אובייקטים חדשים מטיפוס String במספר דרכים. להלן הבולטים שבהם:

public **String**()

© 2007 Zindell Technologies, Ltd.

ליצירת אובייקט שמייצג מחרוזת ריקה ("").

public **String**(String value)

ליצירת אובייקט שמייצג מחרוזת שזהה למחרוזת שמיוצגת על ידי אובייקט ה-String שה-reference שלו נשלח אל ה-constructor בעת הפעלתו.

public **String**(char[] value)

ליצירת אובייקט String חדש שמתאר את מחרוזת התווים אשר מאוחסנת במערך של chars, שאת ה-reference שלו ה-constructor מקבל.

public **String**(char[] value, int offset, int count)

ליצירת אובייקט String חדש שמתאר את מחרוזת התווים אשר מאוחסנת במערך שה-reference שלו נשלח אל ה-constructor בצירוף ציון מספר האינדקס במערך (offset) שהחל ממנו ייקחו count תווים ליצירת המחרוזת.

מן הרגע שבו נוצר אובייקט חדש מטיפוס String לא ניתן לשנות את המחרוזת שהוא מייצג. יש לשים לב, שעם זאת, עדיין ניתן לשנות את ה-reference שמוחזק במשתנה שמצביע על אובייקט ה-String כך שיצביע על אובייקט אחר מטיפוס המחלקה String (אובייקט אשר ייצג מחרוזת אחרת).

התכנית הבאה מדגימה שימוש פשוט באובייקט מטיפוס המחלקה String.

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

© 2007 Zindell Technologies, Ltd.

```
{
        int index;
        String str1=new String("Hello");
        String str2="Israel";
        printy(str1,str2);
}

private static void printy(String strFirst, String strSecond)
// notice that the main method is a static function
      // which means that it can treats methods and
// variables that belongs to the class only if
// they are static
{
        System.out.println(strFirst+" "+strSecond);
}
}
```

מתודות שמוגדרות במחלקה - String Methods

במחלקה String מתודות שימושיות רבות. בחלק זה אציג חלק מהן.

public int length()

מחזירה את אורך המחרוזת

© 2007 Zindell Technologies, Ltd.

```
public char charAt(int index)
```

מחזירה את התו שנמצא במחרוזת במקום שמספר ה-index שלו הוא index.

```
public void getChars(int srcBegin,
                     int srcEnd,
                     char[]dest,
                     int destBegin)
```

מעתיקה אוסף של תווים שנמצאים באובייקט ה-String אשר עליו המתודה הופעלה אל מערך מטיפוס char שה-reference שלו מאוחסן בפרמטר dest. ההעתקה מתבצעת החל מתו srcBegin באובייקט שממנו המתודה הופעלה עד לתו שמספר האינדקס שלו srcEnd (ולא כולל אותו). התווים שמועתקים אל המערך מועתקים אליו החל ממקום destBegin.

```
public char[] toCharArray()
```

מתודה זו מחזירה reference למערך של chars אשר מכיל בתוכו את התווים אשר מופיעים במחרוזת שהאובייקט מייצג. המתודה לא מוסיפה '\0' אל סוף מערך התווים שהיא מחזירה.

```
public String substring(int begin)
```

מתודה זו מחזירה reference לאובייקט חדש מטיפוס String שמייצג תת מחרוזת למחרוזת שמייצג האובייקט. תת המחרוזת שמוחזרת מורכבת מכל התווים שהחל מהתו שמספר האינדקס שלו begin ועד לסופה של המחרוזת שמייצג האובייקט.

© 2007 Zindell Technologies, Ltd.

public String substring(int begin, int end)

מתודה זו מחזירה reference לאובייקט חדש מטיפוס String שמייצג תת מחרוזת למחרוזת שמייצג האובייקט. תת המחרוזת שמוחזרת מורכבת מכל התווים החל מהתו שמספר האינדקס שלו begin (כולל) ועד לתו שמספר האינדקס שלו end (לא כולל).

התכנית הבאה מדגימה את השימוש במתודות אלה:

```
class StringMethodsDemo
{
    public static void main(String args[])
    {
        int index,leng;
        char tav, tavs[] = new char[10];
        for (int h=0;h<10;h++)
            tavs[h] = '$';
        String str1 = "Hello";
        String result;
        System.out.println("str1 is : " + str1);
        leng = str1.length();
        System.out.println("The length of str1 is : " + str1.length());
        tav = str1.charAt(0);
```

© 2007 Zindell Technologies, Ltd.

```
        System.out.println("The first char in str1 is : " + tav);

        str1.getChars(0,str1.length(),tavs,0);

        System.out.println("The chars in str1 using getChars are : ");

        for (index=0; index<=str1.length(); index++)

        {

                        System.out.println("tavs["+index+"]="+tavs[index]);

        }

        System.out.println("The chars in str1 using toCharArray are : ");

        tavs = str1.toCharArray();

        for (index=0; index<tavs.length; index++)

        {

                System.out.println("tavs["+index+"]="+tavs[index]);

        }



        result = str1.substring(3);

        System.out.println("The chars starting in place 3 are : " + result);

        result = str1.substring(1,3);

        System.out.println("The chars from place 1 to 3(excluded) are : " + result);

    }

}
```

© 2007 Zindell Technologies, Ltd.

השוואה בין מחרוזות – String Comparison

אובייקט מטיפוס המחלקה String הוא כמו כל אובייקט אחר. מסיבה זו, השוואה בין שתי מחרוזות שמיוצגות על ידי שני אובייקטים מטיפוס String באמצעות אופרטור ההשוואה, ==, עלולה לתת תוצאות שגויות.

לדוגמא, בהנחה שהמשתנים str1 ו- str2 מכילים references לשני אובייקטים מטיפוס String, השוואה ביניהם באמצעות אופרטור ההשוואה תחזיר true רק אם שני ה-references זהים. מסיבה זו, גם אם שתי המחרוזות זהות, אך הן נמצאות במקומות אחרים בזיכרון ההשוואה הזו תחזיר false.

```
String s1, s2;
.
.
.
if (s1==s2)
{
    . . .
}
```

השוואה כגון זו תצליח רק אם שני המשתנים מכילים את אותו reference.

עם זאת, כדאי לשים לב לכך שכאשר שתי מחרוזות תווים זהות נוצרות מבלי לעשות שימוש באופרטור new, כי אם באופן מפורש על ידי כתיבת המחרוזת מתוחמת בגרשיים כפולים, הן ייווצרו באותו מקום בזיכרון. מסיבה זו, בדוגמא הבאה יודפס "HELLO":

```
String str1=”abc”;
```

© 2007 Zindell Technologies, Ltd.

```
String str2="abc";
if (str1==str2)
    System.out.println("HELLO");
```

המחלקה String מספקת מספר מתודות לביצוע השוואה בין אובייקט מטיפוס String מסויים לאובייקט אחר מטיפוס String.

public boolean equals(String otherString)

מחזירה true אם המחרוזת שמיוצגת על ידי otherString זהה למחרוזת שמיוצגת על ידי האובייקט שעליו מתודה זו הופעלה.

public boolean equalsInoreCase(String otherString)

מחזירה true אם המחרוזת שמיוצגת על ידי otherString זהה למחרוזת שמיוצגת על ידי האובייקט שעליו מתודה זו הופעלה. בביצוע ההשוואה אין התחשבות בהבדלים שבין אותיות קטנות וגדולות.

public int compareTo(String otherString)

מחזירה 0 אם שתי המחרוזות זהות, משמע: ערכן זהה. (הערך מחושב על פי ערך התווים בחישוב עפ"י טבלת ה-Unicode.

מחזירה ערך שלילי אם המחרוזת הנתונה בעלת ערך לקסיקוגרפי נמוך מהאחרת.

מחזירה ערך חיובי אם המחרוזת הנתונה בעלת ערך לקסיקוגרפי גבוה מהאחרת.

© 2007 Zindell Technologies, Ltd.

public boolean startsWith(String prefix)

מחזירה true אם המחרוזת הנתונה מתחילה במחרוזת האחרת

public boolean endsWith(String suffix)

מחזירה true אם המחרוזת שמיוצגת על ידי האובייקט מסתיימת במחרוזת האחרת (suffix).

```
public boolean regionMatches(int thisBgn,
                             String otherStr,
                             int otherStr,
                             int length)
```

מחזירה true אם שתי המחרוזות זהות בטווח המצוין. ההשוואה נעשית החל ממספר האינדקס thisBgn במחרוזת שמייצג האובייקט לאורך length תווים. במחרוזת האחרת ההשוואה נעשית החל מהתו שמספר האינדקס שלו הוא otherStr. מתודה זו מתחשבת בהבדלים של אותיות קטנות/גדולות.

```
public boolean regionMatches(boolean ignoreCase,
                             int thisBgn,
                             String otherStr,
                             int otherStr,
                             int length)
```

מתודה זו פועלת בדומה למתודה הקודמת, אלא שניתן לקבוע בה באמצעות שליחת הערך true בתור הערך הראשון כי ההשוואה תתבצע תוך התעלמות מהבדלים של אותיות קטנות/גדולות.

© 2007 Zindell Technologies, Ltd.

התכנית הבאה מדגימה את השימוש במתודות להשוואה בין מחרוזות:

```
class StringCompareDemo1
{
  public static void main(String args[])
  {
      int index,leng;
      char tav, tavs[] = new char[10];
      String str1 = "Hello Israel";
      String str2 = "hello israel";
      String str3 = "hello";
      String str4 = "zebra";
      String str5 = "rael";
      System.out.println("str1 is : " + str1);
      System.out.println("str2 is : " + str2);
      System.out.println("str3 is : " + str3);
      System.out.println("str4 is : " + str4);
      System.out.println("str5 is : " + str5);
      if (str1.equalsIgnoreCase(str2))
      {
        System.out.println("using equalsIgnoreCase str1 equals ");
```

© 2007 Zindell Technologies, Ltd.

```
  System.out.println("str2 regardless of case");
}
if (str4.compareTo(str3)<0)
{
  System.out.println("using compareTo str3 bigger than str4");
}
else
  if (str4.compareTo(str3)>0)
    System.out.println(
                "using compareTo str4 bigger than str3");
else
  System.out.println("using compareTo str4 equal str3");
if (str2.startsWith(str3))
  System.out.println(
     "using startsWith str2 starts with str3");
if (str1.endsWith(str5))
  System.out.println(
     "using endsWith str1 ends with str5");
if (str1.regionMatches(1,str3,1,4))
{
```

© 2007 Zindell Technologies, Ltd.

```
      System.out.println(
            "using regionMatches str1 and str3 have the ");
      System.out.println("same 4 chars starting at place 1");
    }
    if (str1.regionMatches(true,0,str3,0,5))
    {
      System.out.print("using regionMatches-ignoreCase is true- ");
      System.out.println(
           "str1 and str3 have the same 5 chars starting at index 0");
    }
  }
}
```

מציאת תוים ותת מחרוזות במחרוזת

במחלקה String קיימות מספר מתודות שמשמשות למציאת תו מסוים או תת מחרוזת מסוימת במחרוזת שהאובייקט מייצג. כל אחת מהמתודות מחזירה את מיקום התו או תת המחרוזת המבוקשים במחרוזת עפ"י האינדקס המקובל: מ-0 עד length-1. אם התו או תת המחרוזת המבוקשת לא נמצאים במחרוזת שמייצג האובייקט אז מוחזר הערך –1.

public int indexOf(char ch)

מתודה שמחזירה את מספר האינדקס אשר בו נמצא התו ch במחרוזת שמייצג האובייקט. החיפוש אחרי התו מתחיל מתחילתה של המחרוזת, ומספר האינדקס של התו הראשון, שנמצא זהה לתו ch, מוחזר.

© 2007 Zindell Technologies, Ltd.

public int indexOf(char ch, int begin)

מתודה שמחזירה את מספר האינדקס אשר בו נמצא התו ch במחרוזת שמייצג האובייקט. החיפוש אחריו מתחיל ממספר האינדקס שנשלח אל המתודה, מ- begin, ומספר האינדקס של התו הראשון שנמצא זהה לתו ch מוחזר.

public int lastIndexOf(char ch)

מתודה שמחזירה את מספר האינדקס אשר בו נמצא התו ch במחרוזת שמייצג האובייקט. החיפוש אחרי התו מתחיל מסופה של המחרוזת, והמיקום של התו הראשון שנמצא זהה לתו ch מוחזר.

public int lastIndexOf(char ch, int fromIndex)

מתודה זו מחפשת את התו ch במחרוזת שממנה היא מופעלת החל מאינדקס fromIndex לכיוון התחלתה. התו הראשון שהמתודה מוצאת כזהה לתו ch מוחזר על ידה.

public int indexOf(String str)

מתודה זו מחפשת מתחילתה של המחרוזת שהאובייקט מייצג את המחרוזת str. מספר האינדקס שבו היא נמצאת מוחזר על ידי המתודה. אם המחרוזת str נמצאת במחרוזת שהאובייקט מייצג יותר מפעם אחת אז המתודה מחזירה את מספר האינדקס של המקום הראשון שבו היא מופיעה, כאשר החיפוש מתחיל מהתחלתה של המחרוזת.

public int indexOf(String str, int begin)

מתודה זו מחפשת במחרוזת שהאובייקט מייצג את המחרוזת str החל מהמקום שמספר האינדקס שלו הוא begin לכיוון סופה. מספר האינדקס שבו היא נמצאת לראשונה מוחזר על ידי המתודה.

© 2007 Zindell Technologies, Ltd.

public int lastIndexOf(String str)

מתודה זו מחפשת במחרוזת שמייצג האובייקט את המחרוזת שמיוצגת על ידי str. החיפוש מתחיל מסופה של המחרוזת לכיוון התחלתה. מספר האינדקס שהחל ממנו str נמצאת לראשונה מוחזר על ידי המתודה.

public int lastIndexOf(String str, int fromIndex)

מתודה זו מחפשת במחרוזת שמייצג האובייקט את המחרוזת שמיוצגת על ידי str. החיפוש מתחיל ממיקום fromIndex לכיוון התחלתה של המחרוזת. מספר האינדקס שהחל ממנו str נמצאת לראשונה מוחזר על ידי המתודה.

הדוגמא הבאה מדגימה את השימוש במתודות אלה:

```
class StringSearchDemo
{
	public static void main(String args[])
	{
		int index,leng;
		char tav, tavs[] = new char[10];
		String str1 = "Hello Israel";
		String str2 = "hello israel";
		String str3 = "hello big israel and hello jerusalem";
		String str4 = "zebra";
		String str5 = "rael";
		System.out.println("str1 is : " + str1);
```

© 2007 Zindell Technologies, Ltd.

```
System.out.println("str2 is : " + str2);
System.out.println("str3 is : " + str3);
System.out.println("str4 is : " + str4);
System.out.println("str5 is : " + str5);
index = str1.indexOf('e');
System.out.println("the first e in str1 is in index : " + index);
index = str1.indexOf('e',4);
System.out.println("the first e in str1 starting in place 4 is in" + " index : " + index);
index = str1.lastIndexOf('e');
System.out.println("the last e in str1 is in index :" + index);
index = str1.lastIndexOf('e',9);
System.out.println("the last e in str1 starting from place 9 is in"+ " index : " + index);
index = str1.indexOf("ello");
System.out.println("the first \"ello\" in str1 is in index : " + index);
index = str3.indexOf("ello",10);
System.out.println("the first \"ello\" in str3 starting in place 10" + " is in index : " + index);
index = str3.lastIndexOf("ello");
System.out.println("the last \"ello\" in str3 is in index : " + index);
index = str3.lastIndexOf("ello",19);
System.out.println("the last \"ello\" in str3 starting from place 19" + " is in index : " + index);
```

© 2007 Zindell Technologies, Ltd.

```
    }
}
```

התכנית הבאה מהווה הדגמה נוספת של מתודות אלה:

```
class StringSearchDemo2
{
    public static void main(String args[])
    {
        int index=0,result=0;
        String str1 = "Hello Israel You are a beautiful country";
        while(index>=0)
        {
            ++index;
            result++;
            index = str1.indexOf(' ',index+1);
        }

        System.out.println(
                 "The sentence has " + result + " words");
    }
}
```

© 2007 Zindell Technologies, Ltd.

מתודות שימושיות אחרות במחלקה String

חשוב לזכור, כי אף אחת מהמתודות ששייכות למחלקה String, לרבות המתודות שייסקרו בחלק זה, לא משנה את המחרוזת שמיוצגת על ידי האובייקט אשר עליו היא פועלת.

כל אחת מהמתודות שייסקרו בחלק זה מחזירה reference לאובייקט חדש מטיפוס String אשר מהווה תוצאה של שינויים למחרוזת שמייצג האובייקט שעליו המתודה הופעלה.

```
public String replace (char oldChar, char newChar)
```

מתודה זו מחזירה reference לאובייקט חדש מטיפוס String אשר מייצג מחרוזת תווים שמתקבלת מהמחרוזת, שמייצג האובייקט שעליו המתודה הופעלה, לאחר שכל תו שזהה לתו oldChar מוחלף בתו newChar.

```
public String toLowerCase()
```

מתודה זו מחזירה reference לאובייקט חדש מטיפוס String אשר מתאר את מחרוזת תווים שמתקבלת מהמחרוזת שמייצג האובייקט כאשר מחליפים את כל אחת מהאותיות הגדולות לאות קטנה.

```
public String toUpperCase()
```

מתודה זו מחזירה reference לאובייקט חדש מטיפוס String אשר מתאר את מחרוזת התווים שמתקבלת מהמחרוזת שמייצג האובייקט כאשר מחליפים את כל האותיות הקטנות לאותיות גדולות.

```
public String trim()
```

מתודה זו מחזירה reference לאובייקט מטיפוס String שמתאר מחרוזת תווים שמתקבלת מהמחרוזת שמייצג האובייקט לאחר שכל תווי הריווח המיותרים שמופיעים בתחילת/סוף המחרוזת נמחקים.

© 2007 Zindell Technologies, Ltd.

התכנית הבאה מדגימה את פעולתן של מתודות אלו:

```
class StringCreateMethods
{
    public static void main(String args[])
    {
        int index,leng;
        char tav, tavs[] = new char[10];
        String str1 = "H e l l o Israel and Holly Jerusalem";
        String str2;
        System.out.println("str1 is : " + str1);
        str2 = str1.replace('o','x');
        System.out.println("\nstr2 = str1.replace(\'o\',\'x\')\nstr2 = "  + str2);
        str2 = str1.toLowerCase();
        System.out.println("\nstr2 = str1.toLowerCase()\nstr2 = " + str2);
        str2 = str1.toUpperCase();
        System.out.println("\nstr2 = str1.toUpperCase()\nstr2 = " + str2);
        str2 = str1.trim();
        System.out.println("\nstr2 = str1.trim()\nstr2 = " + str2);
    }
}
```

© 2007 Zindell Technologies, Ltd.

מתודות לחיבור בין מחרוזות

קיימת מתודה לשרשור של מחרוזת נוספת למחרוזת שמייצג האובייקט הנתון, והחזרת ה-reference של האובייקט החדש שמייצג את המחרוזת החדשה.

```
public String concat(String otherString)
```

הפעולה של מתודה זו זהה לפעולתו של האופרטור +, ולפעולתו של האופרטור =+ כאשר הם פועלים על מחרוזות תווים.

לדוגמא:

```
String fName = s1.concat(“.txt”);
```

שורה זו זהה לשורה:

```
String fName = s1 + “.txt”;
```

למרות שב-JAVA לא ניתן להגדיר אופרטורים מחדש כפי שניתן ב- C++, האופרטורים + ו- =+ בהקשר של מחרוזות תווים מוגדרים כבר כחלק מהשפה.

דוגמא נוספת:

```
String str1 += str2;
```

בדוגמא זו, JAVA מחברת את str2 אל סופה של str1 ואת הכתובת של המחרוזת החדשה שמתקבלת מכניסה אל תוך str1, בעקבות פעולה זו, המחרוזת שה-reference שלה הוחזק בתוך המשתנה str1 לא משתנה (ואם reference אליה מוחזק גם במשתנה אחר, אז היא תמשיך להתקיים). מה שהשתנה הוא שנוצרה מחרוזת חדשה וה-reference שלה הוכנס אל תוך str1. הדוגמא הבאה מדגימה פעולות שרשור אלה:

© 2007 Zindell Technologies, Ltd.

```
class ConcatStringDemo
{
    public static void main(String args[])
    {
        int index,leng;
        char tav, tavs[] = new char[10];
        String str0 = "Jerusalem";
        String str1 = "Hello";
        String str2 = "Israel";
        String str3, str4, str5;
        System.out.println("str1 is : " + str1);
        System.out.println("str2 is : " + str2);
        str3 = str1 + " " + str2;
        str4 = str1 + " " + str0;
        str5 = str4.concat(" and ");
        str5 += str3;
        System.out.println(str5);
    }
}
```

© 2007 Zindell Technologies, Ltd.

המרה למחרוזת תווים

במחלקה Object מוגדרת המתודה toString אשר מחזירה ייצוג של מחרוזת תווים לאוביקט שממנו (עליו) היא הופעלה. המתודה toString() מוגדרת באופן כזה שמה שהיא מחזירה הוא מחרוזת תווים שכוללת בתוכה את שם המחלקה בתוספת מספר בבסיס 16.

המתודה println – אם היא מקבלת reference לאובייקט אז היא מפעילה על האובייקט שקיבלה את המתודה toString ומה שהיא מחזירה מודפס למסך. אם לא נגדיר במחלקה שממנה האובייקט נוצר את המתודה toString מחדש, אז תפעל המתודה toString אשר הגיעה בהורשה מהמחלקה Object, ומה שיודפס יהיה שם המחלקה ומספר בבסיס 16 אשר מתקבל באמצעות הפעלת המתודה hashcode על האובייקט האמור. השימוש במתודה toString – לאחר שהוגדרה מחדש כמובן – מתאים במיוחד למטרות debugging. התכנית הבאה מדגימה זאת:

```
class Kuty
{
    int x,y;
}
class Snoopy
{
    int x,y;
    public String toString()
    {
        return "Halleluya";
    }
}
```

© 2007 Zindell Technologies, Ltd.

```
public class ConvertToString
{
        public static void main(String args[])
        {
            Kuty panter = new Kuty();
            Snoopy snap = new Snoopy();
            System.out.println(panter);
            System.out.println(snap);
        }
}
```

במחלקה String קיימת מתודה סטטית ששמה valueOf אשר מקבלת ערך מטיפוס בסיסי כלשהו או reference לאובייקט מטיפוס מחלקה כלשהי, ומחזירה את הייצוג המחרוזתי של מה שהיא קיבלה. אם המתודה valueOf מקבלת reference לאובייקט אז מה שהיא מחזירה זה את הערך שמוחזר מהפעלת המתודה toString על אותו אובייקט.

המחלקה StringBuffer

זוהי מחלקה שדומה למחלקה String. יש בה מתודות שלא קיימות במחלקה String, והשוני הגדול בינה לבין המחלקה String הוא שאת המחרוזת שמייצג אובייקט מטיפוס StringBuffer ניתן לשנות (את המחרוזת שמיוצגת על ידי אובייקט מטיפוס String לא ניתן לשנות). אובייקט מטיפוס StringBuffer שימושי לשינוי ובנייתן של מחרוזות תווים. המתודה append משמשת להוספת מחרוזת תווים למחרוזת שבתוך אובייקט מטיפוס StringBuffer. המתודה הזו מקבלת כארגומנט כל ערך מכל טיפוס.

© 2007 Zindell Technologies, Ltd.

מתודות שימושיות נוספות שקיימות במחלקה StringBuffer הן:

public void insert(int offset, char[]set)

מתודה זו מכניסה אל תוך המחרוזת שמיוצגת על ידי אובייקט ה-String Buffer החל ממיקום offset את התווים שמאוחסנים במערך מטיפוס char שנשלח את המתודה insert. המתודה מכניסה את התווים תוך שהיא דוחפת קדימה את התווים הקיימים (התווים המוכנסים לא דורסים את התווים הקיימים. התווים המוכנסים דוחפים קדימה את התווים הקיימים).

public void setCharAt(int index, char ch)

מתודה זו מכניסה אל תוך המחרוזת שמיוצגת על ידי האובייקט את התו ch במקום ה-index.

public void setLength(int length)

מתודה זו חותכת את המחרוזת המיוצגת על ידי האובייקט ומותירה אותה באורך length תווים.

public String toString()

מתודה זו מחזירה reference לאובייקט מטיפוס String אשר מתאר את המחרוזת שמתוארת על ידי אובייקט ה-StringBuffer שממנו המתודה הופעלה.

התכנית הבאה מדגימה את השימוש במתודות אלה:

© 2007 Zindell Technologies, Ltd.

```
class StringBufferDemo
{
	public static void main(String args[])
	{
		int index,leng;
		char tav, tavs[] = new char[10];
		String str1 = "Hello Israel";
		String str2 = "hello israel";
		String str3 = "hello";
		String str4 = "zebra";
		String str5 = "rael";
		System.out.println("str1 is : " + str1);
		System.out.println("str2 is : " + str2);
		System.out.println("str3 is : " + str3);
		System.out.println("str4 is : " + str4);
		System.out.println("str5 is : " + str5);
		StringBuffer struma = new StringBuffer();
		struma.append(str3);
		struma.append(" ");
		struma.append(str4);
```

© 2007 Zindell Technologies, Ltd.

```
            System.out.println(struma);
            char tavim[] = new char[5];
            tavim[0] = 'b';
            tavim[1] = 'a';
            tavim[2] = 'r';
            tavim[3] = 'a';
            tavim[4] = 'k';
            struma.insert(3,tavim);
            System.out.println(struma);
            struma.setLength(2);
            System.out.println(struma);
        }
}
```

כאן המקום לתאר את מה שקורה כאשר הקומפיילר נתקל בשורה כדוגמת:

“abc” + “def” + “ghi”

האופן שבו השורה מטופלת הוא כדלקמן:

(((new StringBuffer(“abc”)).append(“def”)).append(“ghi”)).toString()

השימוש ב-StringBuffer יותר יעיל מאשר להשתמש באופרטור =+ שכבר מוגדר כאופרטור שיכול לפעול בין Strings

© 2007 Zindell Technologies, Ltd.

כיוון שבפעולתו של האחרון יש פעילות רבה יותר של ה-CPU בכל הקשור ליצירתם של אובייקטים חדשים והקצאת זיכרון.

המחלקה StringTokenizer

מחלקה זו משמשת להפרדת מחרוזת נתונה למספר מחרוזות על פי תו/מחרוזת מפריד/ה אשר ניתן/נת לקביעה מראש בעת יצירתו של אובייקט מטיפוס StringTokenizer.

ה-constructors אשר מוגדרים במחלקה זו כוללים את ה-constructors הבאים:

```
public StringTokenizer(String str, String delim, boolean returnDelims)
```

ליצירת אובייקט מטיפוס StringTokenizer אשר יאפשר את חלוקת המחרוזת str למחרוזות קטנות בהתאם לתו/מחרוזת המפריד/ה delim. אם returnDelims הוא true אז כאשר תופעל המתודה nextToken המחרוזת שתוחזר תכלול בתוכה גם את התו המפריד delim.

```
public StringTokenizer(String str, String delim)
```

ליצירת אובייקט מטיפוס StringTokenizer אשר יאפשר את חלוקת המחרוזת str למחרוזות קטנות בהתאם לתו/מחרוזת המפריד/ה delim.

```
public StringTokenizer(String str)
```

ליצירת אובייקט מטיפוס StringTokenizer אשר יאפשר את חלוקת המחרוזת str למחרוזות קטנות כשהתו המפריד הוא אחד התוים הבאים: \f\ \r\ \n\ t.

בין המתודות אשר מוגדרות במחלקה זו ניתן לזהות את המתודות הבאות:

```
public String nextToken()
```

מתודה זו מחזירה את תת המחרוזת הבאה על פי התו המפריד שנקבע בעת יצירת האובייקט.

© 2007 Zindell Technologies, Ltd.

```
public boolean hasMoreTokens()
```

מתודה זו מחזירה true אם קיימות עוד תת מחרוזות שטרם הוחזרו על ידי המתודה nextToken.

העברת ארגומנטים אל המתודה main

כל מחלקה שמהווה אפליקציה חייבת לכלול את המתודה main כשהיא מוגדרת באופן הבא:

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

מתודה זו מקבלת reference למערך של Strings אשר מכיל בכל אחד מתאיו reference לאובייקט מטיפוס String. ה-references שמועברים הם של כל אחת ממחרוזות התווים שנרשמות בשורת הפקודה אחרי שם המחלקה שמריצים. הריווח הלבן שמופיע בין כל מילה ומילה הוא זה שגורם לכך שכל מילה – נוצר לשם ייצוגה אובייקט מטיפוס String וה-reference שלו מועבר בהתאם כאיבר במערך שמועבר אל המתודה main. אם בשורת הפקודה, אחרי פקודת ההרצה של התכנית רושמים מחרוזת של תווים מתוחמת בגרשיים כפולים אז כל המחרוזת כל כולה – יווצר אובייקט מטיפוס String אשר ייצג אותה, וה- reference שלו יועבר אל המתודה main בתוך המערך שמועבר ונקלט בפרמטר args.

שם הפרמטר שקולט לתוכו את ה-reference למערך ה-Strings הוא args. טכנית ניתן גם לקבוע שם אחר. זהו השם המקובל בלבד.

כדאי לשים לב לכך שלהבדיל מ-C, ב-JAVA שם התכנית שמורצת לא מועבר במערך ה-Strings שמועבר אל main.

© 2007 Zindell Technologies, Ltd.

התכנית הבאה מדגימה את האפשרות להעברת ערכים אל המתודה main. את התכנית יש להריץ משורת הפקודה של DOS, ולרשום אחרי השם של התכנית מחרוזות תווים כשריווחים מפרידים בין כל אחת מהן.

```
class StringsToMain

{

        public static void main(String args[])

        {

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

                {

                        System.out.println("Argument number " + (i+1) + " is : " + args[i]);

                }

                for (int i=args.length-1; i>=0; i--)

                {

                        System.out.println(

                                        "Argument number " + (i+1) + " is :" + args[i]);

                }

        }

}
```

© 2007 Zindell Technologies, Ltd.