מחלקות (Classes)

הקדמה לתכנות מונחה עצמים - Introduction

מחלקות, אובייקטים ומשתנים - Classes, Objects, Variables

אופן השימוש במחלקות - Using Java Classes

אופן הגדרתה של מחלקה - Class Declaration

הרשאות גישה - Access Modifiers

מתודות חופפות - Overloading Methods

מתודות בונות - Constructors

משתנים סטטיים - Static Variables

מתודות סטטיות - Static Methods

סיום חייו של אובייקט – Finalization

שמירת ההגדרה של המחלקה בתוך קובץ - Class Definition Source Code

יצירת קבועים - Final Variables

תכנון נכון של המחלקה - Class Design Hints

© 2007 Zindell Technologies, Ltd.

הקדמה לתכנות מונחה עצמים

בהביטנו סביב ניתן לזהות אובייקטים ללא הפסק. חלק מהאובייקטים הם אובייקטים פיזיים ומוחשים (כיסא, מחשב, שולחן..), וחלק מהם אובייקטים מופשטים (נסיעה לחו"ל, משחק כדור-סל, ריאיון עבודה...).

בכל אובייקט ניתן לזהות תכונות שמאפיינות אותו, ופעולות שהוא יכול לעשות. משום כך ניתן גם לראות בכל אובייקט "כאילו" יש בו משתנים שמכילים ערכים, ופעולות מסוימות שהוא מסוגל לבצע. לדוגמא: אובייקט שהוא הסטודנט – ניתן לומר כי יש בו למשל את המשתנים הבאים: גובה, גיל, מספר תעודת זהות וכי כל אחד מהם מכיל ערך מסוים. באופן דומה, ניתן גם לומר כי אובייקט שהוא סטודנט יכול לבצע את הפעולות הבאות : ללמוד, לאכול, לישון ולבלות.

אובייקטים שיש להם תכונות ופעולות משותפים ניתן לקבץ לקבוצה. בכל אובייקט ששייך לקבוצה נתונה ניתן יהיה למצוא את כל אחד מהמשתנים שהוגדרו בקבוצה. על כל אחד מהאובייקטים ששייכים לקבוצה נתונה ניתן יהיה להפעיל את כל אחת מהמתודות שהוגדרו בקבוצה.

עבור כל קבוצה של אובייקטים ניתן להגדיר מחלקה (class). ההגדרה של המחלקה כוללת את ההגדרה של המשתנים שניתן יהיה למצוא בכל אחד מהאובייקטים ואת ההגדרה של המתודות שניתן יהיה להפעיל על כל אחד מהם.

ניתן לראות במחלקה שמוגדרת מעין הגדרתה של תבנית, אשר כל אובייקט שיווצר ממנה יכיל את כל אחד מהמשתנים שהוגדרו בתבנית, וכמו כן, ניתן יהיה להפעיל עליו את כל אחת מהמתודות שהוגדרו גם הן בתבנית.

© 2007 Zindell Technologies, Ltd.

שלב ראשון בכתיבת התכנית הוא זיהוי האובייקטים שפועלים בה. לאחר זיהוים יש לנסות לקבוע מהן המחלקות שעלינו להגדיר.

תכנית ב-JAVA היא מערכת של אובייקטים שמקיימים ביניהם אינטראקציה. כל אובייקט נוצר ממחלקה שאליה הוא שייך. בתכנון התכנית יש לקבוע, תחילה, מהן המחלקות שיש להגדיר, ומהם הקשרים שישררו ביניהם. הקשרים שיקבעו בין המחלקות ישרו, למעשה, בין האובייקטים שיווצרו מאותן מחלקות. תחילה יש לקבוע מהן המחלקות הכלליות, ורק אחר כך לגזור (להוריש) מהן את המחלקות היותר ספציפיות. לאחר הגדרת המחלקות הספציפיות, נוכל לייצור מהן אובייקטים, ולכתוב את התכנית בהתאם. נושא ההורשה ודאי אינו מוכר לך, ולכן ודאי תרצה לחזור ולקרוא שורות אלה לאחר הפרק שמוקדש לנושא ההורשה.

אובייקט בנוי, לעתים, כאובייקט שקשור לאובייקטים אחרים. ניתן לדמות מצבים אלה כמצבים שבהם האובייקט מכיל בתוכו אובייקטים אחרים, כדוגמת אובייקט שמתאר מכונית ומכיל בתוכו 4 אובייקטים שמתארים 4 גלגלים. דימוי זה מדויק ב- C++, אך לא ב-JAVA. ב- JAVA, להבדיל מ C++, משתנה מטיפוס מחלקה איננו אובייקט. משתנה מטיפוס מחלקה ב-JAVA הוא משתנה אשר מכיל בתוכו reference של שטח בזיכרון שבו נמצא האובייקט. ה reference שיש לכל אובייקט איננו הכתובת שלו. את הכתובת של כל אובייקט ואובייקט רק ה JVM יודע. ב Java אין לנו כל אפשרות לקבל את המידע הזה ואין לנו כל אפשרות לגשת לכתובות מסויימות בזיכרון. ה JVM מנהל את הזיכרון המוקצה לאובייקטים השונים באופן שמאפשר לו לשנות את שטחי הזיכרון המוקצים תוך שמירת ה-references ללא שינוי. ה JVM מחזיק מעין טבלה ובה בכל שורה מצויינת כתובת בזיכרון של שטח זיכרון שמשמש אובייקט ולצידה ה reference שבאמצעותו ניתן לגשת לאובייקט בתוך התכנית.

תכנות מונחה עצמים מייצג תפיסה תכנותית שונה בתכלית מזו שמייצג התכנות האיטרטיבי. אם אינך מכיר שפת תכנות מונחה עצמים, לימודה של JAVA עבורך יהיה יותר מאשר עוד שפה. תוכנית שכתובה בשפת תכנות מונחה עצמים הרבה יותר גמישה לשינויים, וגם יותר קצרה ואלה רק חלק מיתרונותיו של תכנות מונחה עצמים.

© 2007 Zindell Technologies, Ltd.

מחלקות, אובייקטים ומשתנים - Classes, Objects & Variables

בהגדרה של מחלקה ניתן לקבוע את התכונות שיהיו בכל אחד מהאובייקטים שיווצרו ממנה, וכמו כן גם את כל אחת מהפעולות שכל אחד מהאובייקטים יוכל לבצע.

ה **data members** הוא המונח המקצועי המקובל למשתנה שמוגדר במחלקה, ואמור להופיע בכל אחד מהאובייקטים שנוצרים ממנה (עם ערך שעשוי להיות שונה בכל אובייקט ואובייקט).

ה **function members** הוא המונח המקצועי המקובל למתודה שמוגדרת במחלקה כמתודה שניתן להפעיל על כל אחד מהאובייקטים שנוצרים מהמחלקה, כשבכל אחת מהפעמים שבהן היא מופעלת היא יכולה לעשות שימוש במשתנים שבתוך אותו אובייקט מסוים שעליו היא פועלת.

כדי לייצור אובייקט (מופע – instance) חדש ממחלקה יש לבצע instantiation מאותה מחלקה, כלומר: to instance , וזאת כדי לקבל אובייקט (מופע, instance) חדש מהמחלקה.

לאחר יצירתו של אובייקט (instance) חדש ממחלקה ניתן יהיה לאחסן את ה-reference שלו במשתנה מטיפוס אותה מחלקה. אין כל מגבלה לאחסן את ה-reference לאובייקט (ה- instance) שנוצר ביותר ממשתנה אחד. בהמשך נראה כי ניתן גם לאחסן את ה reference במשתנה מטיפוס מחלקה שהורישה למחלקה שממנה נוצר האובייקט או במשתנה מטיפוס interface שמיושם במחלקה שממנה האובייקט נוצר. בשלב זה, כיוון שעדיין לא הכרנו את נושא ההורשה, לא נתעמק בסוגייה.

ניתן לראות בהגדרתה של מחלקה ב-JAVA את הסביבה ואת אופן היישום של כל אחד מהאובייקטים שיווצרו ממנה.

© 2007 Zindell Technologies, Ltd.

אופן השימוש במחלקות - Using Java Classes

כדי לייצור אובייקט (מופע, instance) חדש ממחלקה מסוימת יש להשתמש במילה השמורה new:

```
new  ClassName (  ) ;
```

המילה השמורה new (למעשה, זהו אופרטור) משמשת ליצירת אובייקט חדש ממחלקה ששמה *ClassName*.

לדוגמא:

```
new Box();
```

פקודה זו תיצור אובייקט חדש מטיפוס המחלקה Box.

בהמשך נראה שאם הוגדר במחלקה constructor או constructors אז ניתן יהיה לייצור את האובייקט באמצעות שליחת ערכים אל ה constructor, ובדרך זו לקבוע כבר בעת יצירת האובייקט את הערכים במשתניו.

דוגמא:

```
new ClassName ( arguments ) ;
```

למעט מערך ומחרוזת תווים (שגם נחשבים לאובייקטים), אובייקט חדש של מחלקה ניתן לייצור אך ורק באמצעות הפעלת האופרטור new. אופרטור זה מאתר שטח פנוי בזיכרון עבור האובייקט החדש, מאתחל אותו (אם הופעל constructor שהוגדר כך שערכים יותחלו – נראה זאת בהמשך) ומחזיר את ה-reference כדי שיאוחסן במשתנה. על המשתנה להיות מטיפוס שזהה לטיפוס האובייקט (כלומר, משתנה מטיפוס המחלקה שממנה יוצרים את האובייקט, או, כפי שנראה בהמשך, מטיפוס מחלקה שמורישה למחלקה של האובייקט או משתנה מטיפוס interface שמיושם במחלקה ממנה האובייקט נוצר).

© 2007 Zindell Technologies, Ltd.

אובייקט מטיפוס String ניתן גם לייצור בדרך מקוצרת : בעצם כתיבתה של מחרוזת תווים שתחומה בגרשיים כפולים. לדוגמא: "Israel is great" היא מחרוזת תווים, ולפיכך תיחשב לאובייקט שיש לו כתובת ושניתן לאחסנה במשתנה מטיפוס String.

אובייקט שהוא מערך ניתן גם ליצור בדרך ששונה מדרך יצירתו של האובייקט הרגיל. בפרק שדן במערכים יובא לכך ההסבר.

המונח המקצועי Class Type Variable הוא המקובל למשתנה שהוא מטיפוס מחלקה כלשהי, ושמתעתד להכיל reference לאובייקט. ניתן לראות ב-reference את כתובתו של האובייקט למרות שזוהי איננה כתובתו האמיתית ובכך להקל על הבנת הנושא.

הדרך ליצירתו של משתנה מטיפוס מחלקתי זהה לדרך ליצירת משתנה מטיפוס בסיסי. יש לרשום את שם המחלקה שמטיפוסה רוצים לייצור את המשתנה, ואחריו את שם המשתנה שרוצים לייצור. לדוגמא:

```
Box myBox;
```

בדוגמא זו נוצר משתנה מטיפוס המחלקה Box. במשתנה שהוא מטיפוס מחלקה מסוימת ניתן לאחסן reference (כתובת) של אובייקט מטיפוס המחלקה שמטיפוסה נוצר המשתנה (בהמשך נראה שקיימת גם האפשרות לאחסן reference לאובייקט מטיפוס מחלקה שיורשת מהמחלקה שמטיפוסה נוצר המשתנה). יש לשים לב להבדל לעומת C++: ב C++ משתנה מטיפוס מחלקה היווה כבר אובייקט בפני עצמו. ב-JAVA הוא איננו אובייקט. הוא רק מכיל reference (כתובת) של אובייקט.

ניתן בשורת ההצהרה על המשתנה לשלב גם את יצירתו של אובייקט והכנסת כתובתו (ה reference שלו) אל תוך המשתנה.

© 2007 Zindell Technologies, Ltd.

לדוגמא:

```
Box myBox = new Box();
```

משתנה מטיפוס מחלקה יכול להכיל reference לאובייקט או null.

כאן המקום להדגיש את ההבדל שבין משתנה מטיפוס מחלקתי למשתנה מטיפוס בסיסי. בעוד שהאחרון מכיל ערך, הראשון מכיל את ה-reference לערך (האובייקט). אלה הם שני סוגי המשתנים שקיימים ב-JAVA, וחשוב להבין נקודה זו.

אם יוצרים אובייקט באמצעות האופרטור new, אך לא שמים את ה-reference של שטח הזיכרון שהאופרטור new הקצה והחזיר באף משתנה אז ה- Garbage Collector יבחין שאף אחד לא משתמש באובייקט שהוקצה ומשום כך הוא ישחרר את שטח הזיכרון.

לדוגמא, כתיבת השורה הבאה תגרום להיווצרות אובייקט שבשלב מסויים הזיכרון שמשמש אותו ישוחרר.

```
new Car();
```

אובייקט חדש מטיפוס המחלקה Car יווצר וכיוון שאף משתנה לא מכיל את הכתובת שלו, ה-Garbage Collector יחסלו, כלומר: ישחרר את שטח הזיכרון שמשמש אותו.

ניתן לבצע השמה ממשתנה מטיפוס Class Type אחד אל משתנה אחר מטיפוס אותו Class Type. במקרה כזה, מה שיוכנס אל תוך המשתנה האחר הוא ה refernce שאוחסן בראשון, ובדרך זו שניהם יצביעו למעשה על אותו אובייקט (יצביעו אל אותו שטח זיכרון).

© 2007 Zindell Technologies, Ltd.

לדוגמא:

```
Car familyCar, sportCar;

familyCar = new Car();

sportCar = familyCar;
```

לאחר פקודות אלה המשתנים familyCar ו- sportCar יכילו את אותו reference. כלומר, שינויים שיבוצעו באובייקט באמצעות המשתנה familyCar יבואו לידי ביטוי גם כאשר נשתמש במשתנה sportCar.

יש לשים לב לכך שהשמה בין שני משתנים מטיפוס מחלקתי לא משנה את ערכו של אף אובייקט.

משתנים מטיפוס מחלקתי ניתנים להשוואה בין האחד לשני. השוואה כזו תחזיר true אך ורק אם שני המשתנים מכילים את אותו reference, כלומר: מצביעים על אותו שטח זיכרון. מסיבה זו, גם אם שני המשתנים מצביעים על אובייקטים זהים לחלוטין אך שנמצאים בשטחי זיכרון שונים תוצאת ההשוואה תהיה false, כיון שהכתובות שונות.

לדוגמא:

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

Car car2 = new Car(2, 3, 4);

if (car1==car2)

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

בדוגמא זו, למרות ששני המשתנים מכילים reference לשני אובייקטים זהים, תוצאת התנאי היא false. המילה EQUAL לא תודפס על המסך.

© 2007 Zindell Technologies, Ltd.

ברגע שנוצר אובייקט מטיפוס מחלקה מסוימת, הגישה לכל אחד מערכיו (הערכים שבתוך המשתנים שיש בו מתוקף היותו שייך לאותה מחלקה מסוימת) נעשית באמצעות אופרטור הנקודה. כותבים את שם המשתנה שמכיל את ה reference לאובייקט ומייד אחריו נקודה מפרידה ואחריה את שם המשתנה שאל ערכו רוצים לפנות.

לדוגמא, נניח כי myCar הוא משתנה מטיפוס Car שמכיל reference לאובייקט מטיפוס Car. במחלקה Car הוגדר המשתנה length. במקרה כזה, ניתן יהיה לרשום:

```
myCar.length = 42;
```

בכך לגרום לערכו של המשתנה length באובייקט שכתובתו מאוחסנת ב-myCar להיות שווה ל 42.

שפת התכנות Java היא שפה ידידותית למתכנת, ובתור שכזו, אין בה את האופרטורים המוכרים מ- C: * ו- -> . כמו כן, היא לא מאפשרת לגשת ישירות לכתובות בזיכרון.

קריאה להפעלתה של מתודה (שעל פי ההגדרה של המחלקה ניתנת להפעלה על אובייקט ששייך אליה) הנה, למעשה, קריאה לאובייקט ובקשה ממנו שיפעיל את המתודה. הקריאה להפעלת מתודה על אובייקט מורכבת משם המשתנה שמכיל את ה reference אל האובייקט, נקודה מפרידה ושם המתודה שרוצים להפעיל כשבתוך הסוגריים שלה רושמים את הערכים ששולחים אליה. (יש לשים לב לכך שלאובייקט אין שם. לאובייקט יש אך ורק reference אשר ניתן לאחסון בתוך משתנה מתאים). לדוגמא, נניח כי קיימת מחלקה ששמה Car, ובמחלקה מוגדרת מתודה ששמה numberOfKm אשר מקבלת את מספר הליטרים של דלק שמילאו במכונית ומחזירה את מספר הקילומטרים שהמכונית תוכל לנסוע. הנח שהמתודה מחזירה את מספר הקילומטרים בערך מטיפוס int. במקרה כזה, יצירת אובייקט מטיפוס Car והפעלת המתודה האמורה עליו תיראה כדלקמן:

```
Car  myCar = new Car();

int  distance = myCar.numberOfKm(50);
```

בשורה השניה ניתן לראות הפעלה של המתודה numberOfKm על האובייקט שנוצר ושליחת הערך 50 אליה.

© 2007 Zindell Technologies, Ltd.

כל מחלקה שמגדירים ב-JAVA יורשת באופן אוטומטי את המשתנים והמתודות שהוגדרו במחלקה Object. בפרק שדן בהורשה נסביר באופן מסודר את המשמעות שיש לביצועה של הורשה. כעת נסתפק בהסבר הבא: מחלקה שיורשת ממחלקה אחרת מקבלת ממנה בהורשה את כל המשתנים והמתודות שמוגדרות בה, כלומר: באובייקט מטיפוס המחלקה החדשה ניתן למצוא את כל המשתנים שהוגדרו במחלקה החדשה וגם את כל המשתנים שהוגדרו במחלקה האחרת שממנה המחלקה החדשה יורשת. כמו כן, המתודות שניתן להפעיל על כל אחד מהאובייקטים שנוצרים מהמחלקה החדשה כוללות גם את כל אחת מהמתודות שמוגדרות במחלקה החדשה וגם את כל אחת מהמתודות שמוגדרות במחלקה המורישה. אחת המתודות שמוגדרות במחלקה Object ומועברת בהורשה אל כל מחלקה שנגדיר היא המתודה equals. באמצעות המתודה equals ניתן להשוות בין אובייקט נתון לאובייקט אחר. המתודה equals שמוגדרת במלקה Object מבצעת השוואה פשוטה של ה-reference של האובייקט שעליו היא מופעלת עם ה-reference של האובייקט האחר, שעימו מבוצעת ההשוואה. בפרק שדן בנושא ההורשה נסביר את האפשרות לבצע overriding למתודה שמוגדרת במחלקה המורישה ואז גם נראה כיצד ניתן לבצע overriding למתודה equals כדי שההשוואה שתתבצע תהיה ברת משמעות. כל עוד לא עושים overriding למתודה equals אז המתודה שתפעל היא זו שמוגדרת במחלקה Object ושפעולתה זהה, למעשה, לפעולתו של האופרטור ==.

בדוגמא הבאה ניתן לראות הדגמה לאמור:

```
public class EqualDemo

{

      public static void main(String args[])

      {

            String str1 = "123";

            String str2 = "456";

            Yogly ogly1 = new Yogly();
```

© 2007 Zindell Technologies, Ltd.

```
            Yogly ogly2 = new Yogly();

            if (ogly1.equals(ogly2))

               System.out.println("ogly1 is equal to ogly2");

            if (str1.equals(str2))

               System.out.println("str1 is equal to str2");

      }

}


class Yogly

{

       int x,y;

       public boolean equals(Yogly yo)

       {

           return (x==yo.x && y==yo.y);

       }

}
```

© 2007 Zindell Technologies, Ltd.

הערך null הוא ערך מיוחד שמשמעותו: no object. אם נותנים למשתנה את הערך null אז המשמעות היא שאותו משתנה לא מחזיק באף reference של אף אובייקט.

לדוגמא:

```
Car familyCar;

familyCar = null;
```

משתנה מטיפוס מחלקה ניתן להשוות ל-null, באמצעות האופרטורים: !=, ==.

לדוגמא:

```
if (familyCar==null)

{

.

.

}
```

ניסיון לפנות לערכו של משתנה member של אובייקט באמצעות משתנה מטיפוס מחלקתי שמכיל null (במקום להכיל reference לאובייקט) יגרום ל- run time error ועל המסך תופיע ההודעה שנזרק RuntimeException (הסברים נוספים ראה בפרק 12). הערך null הוא ערך מיוחד שמשמעותו: אין אובייקט והוא יכול להישמר בכל משתנה שיכול להכיל reference לאובייקט.

בתוך instance method ניתן לעשות שימוש במילה השמורה this. בכל רגע נתון בזמן ההרצה, this מכילה reference לאובייקט הנוכחי, האובייקט שעליו הופעלה אותה מתודה. כלומר, ה- reference לאובייקט, שעליו המתודה פועלת.

© 2007 Zindell Technologies, Ltd.

תכנית לדוגמא:

```
class Cord
{

	private int x,y;
	private byte color;

	public void set(int a,int b, byte color)
	{
		this.x=a;
		// the parameter's name is different from
		// the instance variable name so the use
		// of the word this isn't necessary !

		this.y=b;
		// the parameter's name is different from
		// the instance variable name so the use
		// of the word this isn't necessary !
```

© 2007 Zindell Technologies, Ltd.

```
        this.color=color;

        // the parameter has the same name as

        // the instance variable color so the

        // use of the word this is necessary

}

public void set(int a,int b)

{

        x=a;

        y=b;

}


public void set(byte clr)

{

        color=clr;

}


public void set()

{

        x=y=0;

        set((byte)0);
```

© 2007 Zindell Technologies, Ltd.

```
        //calling the function set(byte) to set the color
        //to be zero. you can use the word 'this'
        //but you don't have to !
    }

    public void set(int a)
    {
        x=a;
        y=a;
    }

    public boolean equals(Cord otherCord)
    {
        if (otherCord.x==x && otherCord.y==y)
            return true;
        else
            return false;
    }
}
```

© 2007 Zindell Technologies, Ltd.

```
public class ThisDemo

{

        public static void main(String args[])

        {

                  Cord CordA1;

                  Cord CordA2;

//                byte clr1=(byte)2;

                  CordA1=new Cord();

                  CordA2=new Cord();

//                CordA1.set(clr1);

                  CordA1.set(10);

                  // Now CordA1 is (10,10) in color 2

                  CordA2.set(10,10,(byte)2);

                  // Now CordA2 is (10,10) in color 2 too !

                  if (CordA1.equals(CordA2))

                      System.out.println("\nThe two new Cords equal");

        }


}
```

© 2007 Zindell Technologies, Ltd.

רוב הזמן אין צורך להשתמש במילה השמורה this. מקרים שבהם מתגלה הצורך ב this כוללים בעיקר את שני המצבים הבאים:

1. כאשר מתודה מוגדרת עם פרמטר שזהה בשמו לשמם של משתנה שמוגדר במחלקה. הדרך להבדיל הין השניים היא להוסיף את המילה this לפני השם של המשתנה.

2. כאשר תוך כדי פעולתה של מתודה נתונה רוצים להפעיל מתודה אחרת ורוצים לשלוח אליה את ה-reference של האובייקט שעליו המתודה הראשונה פועלת. במקרה כזה נוכל לשלוח אל המתודה השניה את המילה this.

כאשר מתודה כלשהי מחזירה ערך מטיפוס מחלקה כלשהי, המשמעות המדויקת היא שהמתודה מחזירה, למעשה, reference לאובייקט. ה-reference שמוחזר – ניתן להפעיל באמצעותו מתודות על האובייקט האמור.

לדוגמא:

```
Jframe jf = new JFrame();

jf.getContentPane().add(panel,"Center");
```

בדוגמא זו רואים כיצד באמצעות ה reference שהוחזר על ידי המתודה getContentPane הופעלה המתודה add. המתודה add הופעלה על האובייקט שה-reference שלו הוחזר על ידי המתודה getContentPane.

© 2007 Zindell Technologies, Ltd.

אופן הגדרתה של מחלקה - Class Declaration

תכנית ב-Java היא אוסף של מחלקות. כל שורת קוד בתכנית שאנו כותבים חייבת להשתייך למחלקה שאנו מגדירים. ב-Java אין אפשרות להגדיר משתנים גלובליים שלא משוייכים לאף מחלקה. כמו כן, גם אין אפשרות להגדיר פונקציות גלובליות. כל משתנה וכל פונקציה שאנו מגדירים חייבים להיות שייכים למחלקה מסויימת. בהמשך נראה שניתן להבחין בין משתנים רגילים (instance variables) ומשתנים סטטיים (static variables). משתנים רגילים (instance variables) משוכפלים בכל אחד מהאובייקטים שנוצרים מהמחלקה. משתנים סטטיים הם משתנים שאינם משוכפלים לכל אחד מהאובייקטים ותפקידם להכיל ערך שמאפיין את המחלקה כולה (מעין תחליף למשתנים הגלובליים שמוכרים משפות תכנות כגון C ו- ++C). בהמשך גם נראה שניתן להבחין בין מתודות רגילות ומתודות סטטיות. בעוד שמתודה רגילה פועלת על אובייקט מסויים המתודה הסטטית איננה פועלת על אובייקט מסויים. אנו נראה שמתודה הסטטית מבצעת פעולה כללית שאיננה מתייחסת לאובייקט מסויים ושבמידה מסויימת היא מהווה מעין תחליף לפונקציות הגלובליות שמוכרות משפות תכנות כגון C ו- ++C.

בעת ההגדרה של מחלקה ניתן להוסיף לשורת הכותרת את הרשאת הגישה שאנו רוצים שתהיה לה. כאשר מדובר במחלקה רגילה ניתן להוסיף את הרשאת הגישה public ובכך להגדירה כמחלקה שניתן להשתמש בה מכל מקום או לא להוסיף דבר (ברירת המחדל היא package friendly) ובכך בעצם להגדירה כמחלקה שניתן להשתמש בה כל עוד נמצאים בגבולות ה-package שאליו היא שייכת. כאשר מדובר בהגדרה של מחלקה פנימית קיימת גם האפשרות להגדירה כמחלקה שהיא protected וכמחלקה שהיא private (נראה זאת בהמשך).

בהגדרתה של המחלקה עדיין אין שום אובייקט. הגדרה של מחלקה – כמוה כיצירתו של cookie cutter. הגדרת המחלקה שקולה ליצירתה של תבנית שעל פיה ניתן ליצור עוגיות אחידות, שלכל אחת מהן יש את אותם משתנים וכמו כן, על כל אחת מהן ניתן להפעיל את אותן מתודות. הערכים במשתנים אשר חוזרים על עצמם בכל אחד מהאובייקטים יכולים כמובן להיות שונים.

© 2007 Zindell Technologies, Ltd.

שמות של מחלקות מתחילים תמיד באותיות גדולות (זו מוסכמה).

כל משתנה שמוגדר בתוך מחלקה (למעט משתנה מסוג static) הוא משתנה שישוכפל עבור כל אובייקט. כלומר, הוא יופיע בכל אובייקט שיווצר מהמחלקה. מסיבה זו, כל משתנה (למעט משתנה static שבו נדון בהמשך) שמוגדר בתוך הגדרה של מחלקה קרוי בשם instance variable. ה-instance variable משוכפל מחדש לכל instance (אובייקט) שנוצר מהמחלקה.

כל אחד מה-instance variables מאותחל או במכוון (באמצעות השמה של ערך בשורה בה הוא מוגדר) או בערך ברירת המחדל שנקבע בהתאם לטיפוס שלו. יש לשים לב לכך שמשתנים מקומיים לא מאותחלים באופן אוטומאטי. משתנים מקומיים חייבים לוודא שמושם בהם ערך לפני שנעשה בהם שימוש.

כל משתנה שמוגדר בתוך מחלקה נחשב ל-instance variable עפ"י ברירת המחדל. רק אם צוינה המילה static בשורת ההגדרה שלו אז הוא לא ייחשב ל-instance variable.

התכנית הבאה מדגימה איתחול לפי ברירת המחדל:

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

© 2007 Zindell Technologies, Ltd.

```
        y=b;
    }
    public boolean equals(Point otherPoint)
    {
        if (otherPoint.x==x && otherPoint.y==y)
            return true;
        else
            return false;
    }
}

public class Init1
{
    public static void main(String args[])
    {
        Point pointA1;
        Point pointA2;
        pointA1 = new Point();
        pointA2 = new Point();
        if (pointA1.equals(pointA2))
```

© 2007 Zindell Technologies, Ltd.

```
                System.out.println("\nThe two new points equal");
        }
}
```

התכנית הבאה מדגימה איתחול מכוון:

```
class Point
{
        private int x=0,y=0;
        public void set(int a,int b)
        {
                x=a;
                y=b;
        }
        public boolean equals(Point otherPoint)
        {
                if (otherPoint.x==x && otherPoint.y==y)
                        return true;
                else
                        return false;
        }
```

© 2007 Zindell Technologies, Ltd.

```
}


public class Init2
{
        public static void main(String args[])
        {
                Point pointA1;
                Point pointA2;
                pointA1=new Point();
                pointA2=new Point();
                if (pointA1.equals(pointA2))
                    System.out.println("\nThe two new points equal");
        }
}
```

כל מתודה שמוגדרת בתוך מחלקה הנה (עפ"י ברירת המחדל) מתודת instance, כלומר, מתודה שכאשר קוראים להפעלתה, היא תפעל על אובייקט מסויים. מסיבה זו, תמיד כאשר מפעילים אותה יש להפעילה תוך ציון ה-reference של האובייקט שעליו רוצים להפעילה. בדרך כלל, כדי להפעיל מתודה של אובייקט מסוים יש לרשום את שם המשתנה שמכיל בתוכו reference לאובייקט, נקודה מפרידה, ומייד אחריה את שם המתודה.

לדוגמא:

```
Car myCar = new Car();
myCar.israelCar();
```

© 2007 Zindell Technologies, Ltd.

זוהי קריאה להפעלת instance method אשר תשנה את האובייקט שעליו היא הופעלה.

כל method מוגדר בתוך מחלקה. כל method שייך לאיזושהי מחלקה. ב-JAVA אין אפשרות לייצור פונקציה שלא תהיה שייכת לאף מחלקה (כמו שאפשר ב-C). כמו כן, ב-JAVA אין אפשרות לייצור מתודה שתהיה שייכת לכל המחלקות.

כל מתודה שמפעילים פועלת על הערכים שבמשתנים שיש באובייקט שעליו היא הופעלה. התייחסות בתוך הגדרת המתודה למשתנים שהוגדרו במחלקה בתור instance variables כמוה כהתייחסות לערכים, שבתוך אותם משתנים, בתוך אובייקט שעליו המתודה הופעלה. לדוגמא:

```
public class Box
{
     int x,y;
     public int area()
     {
            return x*y;
     }
}
```

כדאי לשים לב למודולריות שקיימת ב-JAVA: כאשר מוסיפים הגדרה של method למחלקה נתונה אין צורך לעשות קומפילציה מחדש לכל המחלקות בתכנית. די בביצועה של קומפילציה למחלקה ששונתה בלבד.

כל מתודה מוגדרת באופן מלא בתוך גבולות ההגדרה של המחלקה. כל הגוף של המתודה מופיע מייד לאחר שורת הכותרת, ושניהם יחד תחומים בתוך הסוגריים המסולסלות של הגדרת המחלקה.

© 2007 Zindell Technologies, Ltd.

בשפת התכנות Java להבדיל משפת התכנות C++:

א. האופרטור :: לא קיים.

ב. כל מתודה שייכת למחלקה מסוימת. לא ניתן לייצור מתודה ששייכת לכל המחלקות או שלא שייכת לאף מחלקה.

ג. לא ניתן להצהיר על מתודה בתוך תחום ההגדרה של המחלקה, ולהגדיר את המתודה מחוץ לתחום ההגדרה של המחלקה, כפי שניתן ב-C++ (ב C++ ניתן למקם את שורת ההצהרה על המתודה בתוך תחום ההגדרה של מחלקה, ואת גוף המתודה להגדיר מחוץ לתחום ההגדרה).

© 2007 Zindell Technologies, Ltd.

הרשאות הגישה

בתחילת שורת ההגדרה של המחלקה, ולפני המילה השמורה class ניתן להוסיף מילה שמורה אשר תקבע את הרשאת הגישה שתהיה למחלקה הנוצרת. האפשרויות הן:

public

קביעת הרשאת הגישה public בתור הרשאת הגישה של מחלקה תגרום לכך שניתן יהיה לעשות שימוש במחלקה הזו בכל מקום.

private

הגדרתה של מחלקה עם הרשאת הגישה private אפשרית רק כאשר מדובר במחלקה פנימית. כאשר מחלקה פנימית (יוסבר בהמשך) מוגדרת עם הרשאת הגישה private אז השימוש בה אפשרי רק בגבולות המחלקה שבה המחלקה הפנימית הוגדרה (יוסבר בהמשך בפרק שדן במחלקות פנימיות).

protected

אפשרות זו תוסבר בפרק שדן בנושא הורשה. אפשרות זו קיימת רק כאשר מדובר במחלקה פנימית.

package friendly

אם לא רושמים שום דבר לפני המילה class אז המחלקה תהיה package-friendly, משמע: ניתן יהיה להשתמש במחלקה רק כל עוד זה ייעשה בגבולות של מחלקה אשר שייכת לאותו package שאליו המחלקה האמורה שייכת. אם בקובץ הגדרת המחלקה לא נרשם משפט ה-package, אז המשמעות היא שהמחלקה שייכת ל Default Package. במקרה כזה, כל המחלקות ששייכות ל-Default Package יוכלו להשתמש בה.

ציון הרשאת הגישה אפשרי גם לכל אחד מרכיבי המחלקה. ניתן להוסיף הרשאת גישה גם לפני משתנה , וגם לפני מתודה. כך למשל, נרצה לעתים להגדיר מתודות שיהיו עם הרשאת הגישה private כאשר השרות שהן נותנות ייחודי למחלקה המסויימת שבה הוא פועל ואנו לא מעוניינים לאפשר את הפעלתן באופן ישיר ממחלקות אחרות.

© 2007 Zindell Technologies, Ltd.

האפשרויות במתן הרשאת גישה למתודות ולמשתנים ומשמעותן:

private

מתודה, שהרשאת הגישה שלה private, ניתן יהיה להפעיל אך ורק בתוך גבולות המחלקה. באופן דומה, משתנה עם הרשאת הגישה private יהיה נגיש באופן ישיר אך ורק אם שורת הקוד שפונה אליו תיכתב בגבולות המחלקה שבה הוא הוגדר.

public

מתודה שהרשאת הגישה שלה public ניתן להפעיל מכל מקום. באופן דומה, משתנה שמוגר עם הרשאת הגישה public נגיש באופן ישיר מכל מקום.

protected

(נראה בהמשך בפרק שדן בהורשה)

package friendly

אם לא רושמים שום דבר לפני הגדרת המתודה אז המתודה תיחשב ל- package-friendly, משמע: ניתן לקרוא להפעלתה רק כל עוד נמצאים בגבולותיה של מחלקה אשר שייכת לאותו package שאליו שייכת המחלקה שבה הוגדרה המתודה האמורה. כאשר מגדירים משתנה כ- package friendly אז ניתן יהיה לפנות אליו באופן ישיר רק אם שורת הקוד שמבצעת זו תיכתב בגבולותיה של מחלקה ששייכת לאותו package שאליו שייכת המחלקה שבה הוגדר המשתנה האמור.

ככלל, יש להשתדל להעניק את הרשאת הגישה private למספר רב ככל האפשר של משתנים ששייכים למחלקה. במתן הרשאת הגישה private למשתנים שמכילים ערכים רגישים של המחלקה אפשר יהיה להבטיח שרק מתודות של המחלקה יוכלו לשנות את ערכם. כמו כן, כתיבת מתודות מיוחדות שיאפשרו גישה למשתנים אלה תיאלץ מתכנתים לגשת אליהם אך ורק דרך אותן מתודות, והיא לא תאפשר את הגישה הישירה אליהם. בדרך זו, ניתן יהיה לוודא שאותם משתנים רגישים מכילים כל העת ערכים חוקיים בלבד (חוקיים מבחינת תכנון המחלקה). לעתים נגדיר מתודה כ-private. אנו נעשה זאת כאשר המתודה מבצעת פעולה שאנו רוצים לאפשר את הפעלתה הישירה מגבולות המחלקה בלבד.

© 2007 Zindell Technologies, Ltd.

מתודות חופפות - Method Overloading

בתוך מחלקה ניתן להגדיר את אותה מתודה במספר גרסאות כשמה שמבדיל ביניהן הוא מספר הפרמטרים ו/או סוגם. כדי שניתן יהיה לכתוב מספר מתודות בשם זהה חייב להיות שוני או במספר הפרמטרים או בטיפוס הערך שלהם. מתודה ששמה זהה לשמה של מתודה אחרת שכבר קיימת במחלקה נקראת בשם: overloaded method.

התכנית הבאה מדגימה מתודות חופפות (המתודה set מוגדרת מספר פעמים במספר גרסאות שונות):

```
class Nkuda
{
        private int x,y;
        private byte color;
        public void set(int a,int b, byte clr)
        {
                x=a;
                y=b;
                color=clr;
        }
        public void set(int a,int b)
        {
                x=a;
                y=b;
```

© 2007 Zindell Technologies, Ltd.

```
}
public void set(byte clr)
{
        color=clr;
}
public void set()
{
        x=0;
        y=0;
        color=0;
}
public void set(int a)
{
        x=a;
        y=a;
}
public boolean equals(Nkuda othernkuda)
{
        if (othernkuda.x==x && othernkuda.y==y)
                return true;
```

© 2007 Zindell Technologies, Ltd.

```
                else
                        return false;
        }
}


public class OverloadingDemo
{
        public static void main(String args[])
        {
                Nkuda nkudaA1;
                Nkuda nkudaA2;
                byte clr1=2;
                nkudaA1=new Nkuda();
                nkudaA2=new Nkuda();
                nkudaA1.set(clr1);
                nkudaA1.set(10);
                // Now nkudaA1 is (10,10) in color 2
                nkudaA2.set(10,10,(byte)2);
                // Now nkudaA2 is (10,10) in color 2 too !
```

© 2007 Zindell Technologies, Ltd.

```
            if (nkudaA1.equals(nkudaA2))
                System.out.println("\nThe two new nkudas equal");
        }
}
```

ניסיון להגדיר שתי מתודות זהות, למעט טיפוס הערך המוחזר שלהן לא מספיק, וגם איננו אפשרי. כאשר יש קריאה להפעלת מתודה שהיא אחת מקבוצת המתודות בעלות אותו שם, הקומפיילר יבחר להפעיל את המתודה שמתאימה ביותר להפעלה על פי מספר הארגומנטים וטיפוס הערך של כל אחד מהם. אם שולחים להפעלת מתודה מסוימת ארגומנט מטיפוס ששונה מהטיפוס שהמתודה מצפה לקבל אז במידה ש-casting אפשרי מבלי שיהיה חשש לאיבוד ערך אז ה-casting יתבצע באופן אוטומאטי. אם קיים חשש לאיבוד ערך אז casting לא יתבצע באופן אוטומאטי. במקרה כזה, כדי שההפעלה של המתודה תעבור קומפילציה יש לבצע casting באופן יזום.

© 2007 Zindell Technologies, Ltd.

מתודות בונות - Constructors

ה-constructor הוא מתודה מיוחדת שתפקידה לאתחל אובייקט חדש. שמו של constructor זהה לשם של המחלקה שבה הוא מוגדר. הדרך הטכנית להגדרת constructor היא כדלקמן:

1. מגדירים מתודה ששמה זהה לשם של המחלקה.
2. אסור שהמתודה תכלול את הפקודה return. אפילו לא return void.
3. רשימת פרמטרים במתודה בהחלט תיתכן.

דוגמא:

```
class Box
{
    int width, height;
    Box(int w, int h)
    {
      width = w;
      height = h;
    }
}
```

בדוגמא זו הוגדרה מחלקה ששמה Box, ובתוך הגדרתה הוגדרה מתודה בונה (constructor), אשר מקבלת שני ערכים מספריים ומכניסה אותם לתוך שני המשתנים (width ו- height) באובייקט החדש שנוצר.

© 2007 Zindell Technologies, Ltd.

יש לשים לב לכך שאם לא מגדירים במחלקה אף constructor אז קיים ה-Default Constructor. זהו constructor ללא פרמטרים אשר קיים באופן אוטומטי בכל מחלקה שמגדירים ב-Java כל עוד לא הגדרנו בה constructor כלשהו אחר ביוזמתנו. ה-Default Constructor היא הפונקציה הבונה אשר מופעלת כאשר יוצרים אובייקט חדש ממחלקה שלא כוללת בהגדרתה אף הגדרה של אף constructor.

ברגע שמוסיפים הגדרה של constructor להגדרת המחלקה, ה-Default Constructor לא קיים יותר. מסיבה זו אם מגדירים במסגרת הגדרת המחלקה constructor אשר מקבל ערך אחד או יותר בהפעלתו, ורוצים שניתן יהיה לייצור אובייקט חדש מבלי לשלוח ערכים בעת יצירתו אז יש להגדיר constructor נוסף אשר לא מקבל אף ערך בעת הפעלתו.

בהגדרתה של מחלקה ניתן לכלול יותר מ-constructor אחד, ובלבד שכל אחד מה-constructors ייבדל מה-constructors האחרים במספר ו/או בטיפוס ערכם של הפרמטרים. זאת ניתן לביצוע, כיוון שה-constructors הן מתודות, וכפי שכבר הוסבר, ניתן לכתוב מתודות חופפות.

בדוגמה הבאה ניתן לראות את הגדרתה של מחלקה אשר כוללת בתוכה הגדרות של מספר constructors (אלה הן, למעשה, מתודות חופפות). תוך כדי פעולתו של constructor ניתן להפעיל constructor אחר של אותה מחלקה באמצעות המילה this. אם בתוך הגוף של constructor רושמים את המילה this בצירוף סוגריים כשבתוכם ערכים – גורמים בכך להפעלתו של constructor אחר של אותה מחלקה, אשר בפעולתו יפעל על האובייקט שנוצר, ולאחר שיסיים את פעולתו יתר הפקודות ב-constructor הנתון יתבצעו. קריאה להפעלת constructor אחר באמצעות this חייבת להופיע בשורה הראשונה של ה-constructor שנתון.

הדוגמא הבאה מציגה מחלקה שכוללת מספר constructors.

© 2007 Zindell Technologies, Ltd.

```
class Tree
{
       int length, color;
       Tree()
       {
            this(10, 4);
       }
       Tree(int leng)
       {
            this(leng, 4);
       }
       Tree(int leng, int col)
       {
            color = col;
            length=leng;
       }
       . . .
}
```

בדוגמא ניתן לראות הגדרה של מחלקה אשר כוללת בתוכה מספר constructors. בתוך ארבעה מבין ה-constructors ניתן לראות הפעלה של constructor אחר באמצעות המילה השמורה this. ניסיון לקרוא תוך כדי פעולתו של constructor מסוים להפעלתו של constructor אחר באמצעות כתיבת שם המחלקה במקום המילה this יגרום לבלבול

© 2007 Zindell Technologies, Ltd.

אצל הקומפיילר: הוא עלול לחשוב שעליו לייצור אובייקט נוסף, וזאת כאשר מה שאנו בעצם רוצים זה לבצע פעולות מסוימות על האובייקט שכעת נוצר. מסיבה זו, כדי להפעיל constructor אחר תוך כדי פעולתו של constructor מסוים יש להשתמש ב-this. הדוגמא הבאה מדגימה את שנאמר עד כה.

```
class Point
{
        private int x,y;
        public Point()
        {
           this(0,0);
        }
        public Point(int a, int b)
        {
           x = a;
           y = b;
        }
        public Point(int a)
        {
                this(a,a);
        }
}
```

© 2007 Zindell Technologies, Ltd.

אתחול ה-instance variables בערכי ברירת המחדל עפ"י טיפוס הערך שלהם נעשה באופן אוטומאטי בעת יצירת האובייקט. לאחר איתחולם מופעל ה-constructor. בתוך ה-constructor אנו נרשום רק פעולות נוספות אשר נרצה שיתבצעו באובייקט החדש. כך למשל, אם נרצה לאתחל את האובייקט החדש בערכים שונים מערכי ברירת המחדל אנו נכתוב זאת בתוך ה-constructor.

© 2007 Zindell Technologies, Ltd.

משתנים סטטיים (משתני מחלקה) - static variables (class variables)

משתנים סטטיים שמוגדרים בתוך מחלקה נקראים גם משתנים מחלקה (class variables). אלה הם משתנים שנוצרים פעם אחת בלבד (הם לא נוצרים שוב ושוב בכל אובייקט חדש), והם באים לתאר את המחלקה כולה או משהו שמשותף לכל אחד מהאובייקטים. כדי שמשתנה שמוגדר במחלקה ייחשב למשתנה סטטי יש צורך להוסיף לתחילת השורה, שבה הוא מוגדר, את המילה static.

לדוגמא:

```
public class Car
{
	double speed, size;
	static int numOfCars = 0;
	public Car(double sp, double sz)
	{
		speed = sp;
		size = sz;
		numOfCars++;
	}
}
```

בדוגמא זו הוגדר במחלקה Car משתנה סטטי ששמו numOfCars. את המשתנה הזה כל האובייקטים שנוצרים מהמחלקה יכולים לראות (משמע, יכולים לגשת אליו ולראות את ערכו). בדוגמא זו המשתנה numOfCars נוצר פעם אחת ויחידה. הוא לא נוצר שוב ושוב בכל אובייקט חדש שנוצר. המשתנה numOfCars מכיל את מספר האובייקטים

© 2007 Zindell Technologies, Ltd.

שנוצרו מהמחלקה Car מהרגע שבו התכנית התחילה לפעול, והוא מתעדכן תוך כדי פעולתו של ה-constructor. כל פעם כשנוצר אובייקט חדש, ערכו של המשתנה הסטטי numOfCars גדל ב-1.

את המשתנה הסטטי כל האובייקטים יכולים לראות. הוא גלוי לכולם. כל האובייקטים – בפנייתם למשתנה הסטטי פונים, למעשה, לאותו שטח בזיכרון. המשתנה הסטטי לא נמצא בתוך כל אחד מהאובייקטים שנוצרים. שטח הזיכרון שמשמש את המשתנה הסטטי מוקצה באופן אוטומטי (גם אם לא נוצר אף אובייקט מהמחלקה) מייד עם הרצתה של התכנית, ומייד אחר כך הוא גם מאותחל בערך ברירת המחדל שבהתאם לטיפוסו.

הגישה למשתנה הסטטי מחוץ למחלקה (כלומר: כאשר לא נמצאים בתוך מתודה ששייכת למחלקה) אפשרית מכל אחד מהאובייקטים שנוצרו מהמחלקה, וגם מהמחלקה עצמה. כדי לפנות אל המשתנה הסטטי יש לרשום את שם המשתנה (שמכיל reference לאובייקט מאותה מחלקה) או את שם המחלקה ואחריהם את שם המשתנה הסטטי כשמה שמפריד ביניהם היא נקודה מפרידה. לדוגמא (המשך הדוגמא הקודמת):

```
Car.numOfCars = 100;
```

או

```
myCar.numOfCars = 100;
```

שתי הדרכים לגישה למשתנה הסטטי אפשריות ומקובלות.

הגישה למשתנה הסטטי בתוך מתודות ששייכות למחלקה נעשית על ידי ציון שמו בלבד, בדומה ל-instance variable.

ניתן לעשות שימוש במשתנה הסטטי בתור קבוע. יצירת קבוע (נציג את הדרך לכך בהמשך הפרק), משמע, יצירת משתנה אשר ערכו לא יכול להשתנות, וקביעת המשתנה כמשתנה static היא דרך מקובלת ביצירת קבועים. יצירת instance variable כקבוע בזבזנית בזיכרון כיוון שבכל אובייקט שנוצר תופס מקום ה instance variable שנוצר כקבוע.

© 2007 Zindell Technologies, Ltd.

אנו ניצור instance variable כקבוע רק כאשר מדובר בקבוע עם ערך שונה וייחודי לכל אובייקט ואובייקט. יצירת משתנה סטטי וקביעתו כקבוע צורכת הרבה פחות זיכרון: רק במקום אחד יישמר ערכו של המשתנה הסטטי.

השימוש במשתנה סטטי במחלקה מקובל כאשר רוצים משתנה אשר מכיל ערך שרלוונטי לכל האובייקטים שנוצרו מהמחלקה, כדוגמת משתנה אשר מכיל את מספר האובייקטים שנוצרו מהמחלקה, או משתנה שמכיל מאפיין משותף לאובייקטים (לדוגמא, משתנה שמכיל את המהירות המירבית שמותרת לכל מכונית. מאפיין זה משותף לכל האובייקטים שנוצרים מהמחלקה Car).

משתנה מסוג class variable (תזכורת: class variable הוא שם נוסף למשתנה סטטי) נוצר ומאותחל מייד עם הרצת התכנית (מייד עם העלאת הגדרתה של המחלקה לזיכרון). המשתנה הסטטי שנוצר מאותחל בערך ברירת המחדל של טיפוס הערך שלו, אלא אם צוין אחרת.

את המשתנים הסטטיים ניתן לאתחל במרוכז באמצעות static initializer block. יש לרשום static, לפתוח סוגריים מסולסלות, ובתוכן לרשום משפטי השמה של ערכים אל תוך משתנים סטטיים. במחלקה ניתן לכתוב יותר static block אחד.

יש לשים לב לכך ש static blocks מתבצעים מייד בתחילת התכנית , ולפני שורות הקוד האחרות. כמו כן, ניתן להכניס ערך למשתנה סטטי יותר מפעם אחת, וגם ב static blocks שונים. במקרה כזה, ה-static blocks יתבצעו לפני יתר התכנית, והם יתבצעו על פי סדר הופעתם. בתכנית הדוגמא הבאה מוצגת פעולתו של ה-static block:

© 2007 Zindell Technologies, Ltd.

```
class Pointy
{
        private int x,y;
        private byte color;
        public static int numOfPoints;
        static
        {
                numOfPoints=1000;
        }
         public Pointy()
        {
                numOfPoints++;
        }
        public void set(int a,int b, byte color)
        {
                this.x=a;
                // the parameter name is different from
                // the instance variable name so the use
                // of the word this isn't neccesary !
                this.y=b;
```

© 2007 Zindell Technologies, Ltd.

```
        // the parameter name is different from
        // the instance variable name so the use
        // of the word this isn't neccesary !
        this.color=color;
        // the parameter has the same name as
        // the instance variable color so the
        // use of the word this is necessery
}
public void set(int a,int b)
{
        x=a;
        y=b;
}

public void set(byte clr)
{
        color=clr;
}
```

© 2007 Zindell Technologies, Ltd.

```
	public void set()
	{
		x=y=0;
		color=0;
	}

	public void set(int a)
	{
		x=a;
		y=a;
	}

	public boolean equals(Pointy otherPoint)
	{
		if (otherPoint.x==x && otherPoint.y==y)
			return true;
		else
			return false;
	}
}
```

© 2007 Zindell Technologies, Ltd.

```
class StaticBlockDemo
{
        public static void main(String args[])
        {
                Pointy pointA1;
                Pointy pointA2;
                pointA1=new Pointy();
                pointA2=new Pointy();
                int num = Pointy.numOfPoints;
                System.out.println("\nThe number of Points that were
                                        created is " + num);
        }
}
```

© 2007 Zindell Technologies, Ltd.

מתודה סטטית - Static(Class) Method

בדומה לקיומם של משתנים סטטיים, כך גם קיימות מתודות סטטיות, אשר קיימות וניתנות להפעלה עוד לפני שבכלל נוצר איזשהו אובייקט (בדומה למשתנה סטטי, שגם הוא ניתן לשימוש עוד לפני שבכלל נוצר איזשהו אובייקט).

כדי לייצור מתודה סטטית יש לכתוב לפני השם של המתודה את המילה השמורה static. את המילה שמתארת את הרשאת הגישה למתודה יש לרשום לפני המילה static.

לדוגמא:

```
public class Car
{
       static int value=3;
       public static void increasValue(int inc)
       {
            value+=inc;
       }
       .
       .
}
```

© 2007 Zindell Technologies, Ltd.

הפעלתה של מתודה סטטית אפשרית (בדומה למשתנה סטטי) בשתי דרכים:

1. פניה להפעלתה באמצעות שם המחלקה.

דוגמא:

```
Car.increaseValue(8);
```

2. פניה להפעלתה באמצעות שם של משתנה שמכיל reference לאובייקט.

דוגמא:

```
myCar.increaseValue(8);
```

במתודות סטטיות לא ניתן לעשות שימוש במילה השמורה this כיוון שלמילה this בתוך גבולות ההגדרה שלהן אין כל משמעות. מתודה סטטית ניתנת להפעלה באמצעות שם המחלקה. במקרה כזה הפעלתן כלל לא נעשית על אובייקט מסוים כך שקיומה של המילה this בתוך מתודה סטטית כלל לא אפשרי.

המתודה main היא דוגמא למתודה סטטית (ששייכת למחלקה כמובן). המתודה main לא שייכת ולא מופעלת על אובייקט מסוים. אם היא לא הייתה static אז לא היה ניתן להפעילה ללא קיומו של אובייקט מהמחלקה שבה היא מוגדרת. בכתיבת המתודה main כמתודה static בתוך המחלקה אנו, למעשה, מאפשרים את הרצת המחלקה כאפליקציה מבלי שנצטרך לייצור מהמחלקה אובייקט. כאשר המתודה main מופעלת לא נוצר עדיין שום אובייקט מהמחלקה, ולכן, אם רוצים לגשת למשתני ה-instance variable של האובייקט חייבים לייצור לשם כך אובייקט מטיפוס אותה מחלקה.

מתודה סטטית לא יכולה להתייחס ל instance variables כיוון שהגישה למתודה הסטטית לא בהכרח נעשית דרך אובייקט מסוים, כך שגם אם היינו רוצים שהגישה ל-instance variables תהיה אפשרית זה לא היה מתאפשר כיוון שמתודה סטטית יכולה לפעול בעקבות הפעלתה באמצעות שם המחלקה ובמקרה כזה אין בכלל instance variables

© 2007 Zindell Technologies, Ltd.

שאפשר להניח שהמתודה הסטטית מתייחסת אליהם.

לעתים מוגדרות מחלקות שכל המשתנים וכל המתודות שלהן מסוג static. במקרים אלו אין כל משמעות ליצירת אובייקט מהמחלקה. מחלקות כאלה משמשות בדרך כלל כמחלקות שמספקות שירותים, כדוגמת שירותי חישוב למיניהם אשר לחישובם אין כל צורך לייצור אובייקט מהמחלקה.

כאשר תגיע לפרק שדן בהורשה תבין את שכעת יאמר בתמציתיות, והוא כי לא ניתן לעשות overriding ל- static method, ובמילים אחרות, לא ניתן להגדיר מתודה סטטית שהגיעה בירושה מחדש. הדוגמא הבאה מציגה את פעולתה של מתודה סטטית:

```
class Kruvit
{
        private int x,y;
        private byte color;
        static int numOfKruvits;
        static
        {
                numOfKruvits=1000;
        }
        static void resetNum()
        {
                numOfKruvits=0;
```

© 2007 Zindell Technologies, Ltd.

```
}
public Kruvit()
{
        numOfKruvits++;
}
public void set(int a,int b, byte color)
{
        x=a;
        y=b;
        this.color=color;
        // the parameter has the same name as
        // the instance variable color so the
        // use of the word this is necessery
}
public void set(int a,int b)
{
        x=a;
        y=b;
}
public void set(byte clr)
```

© 2007 Zindell Technologies, Ltd.

```
{
        color=clr;
}
public void set()
{
        x=y=0;
        color=0;
}
public void set(int a)
{
        x=a;
        y=a;
}
public boolean equals(Kruvit otherKruvit)
{
        if (otherKruvit.x==x && otherKruvit.y==y)
                return true;
        else
                return false;
}
```

© 2007 Zindell Technologies, Ltd.

```
}


class StaticMethodDemo

{

        static public void main(String argv[])

        {

                Kruvit KruvitA1;

                Kruvit KruvitA2;

                KruvitA1=new Kruvit();

                KruvitA2=new Kruvit();

                System.out.println("\nThe number of Kruvits that were
                     created is " + Kruvit.numOfKruvits);

                Kruvit.resetNum();        // Calling a class method

                System.out.println("\nThe number of Kruvits that were
                     created is " + Kruvit.numOfKruvits);

        }

}
```

© 2007 Zindell Technologies, Ltd.

סיום חייו של האובייקט - Finalization

במחלקה Object מוגדרת המתודה finalize(). המתודה הזו (בגרסתה הכתובה במחלקה Object) לא מבצעת שום פעולה מיוחדת, אך היא מופעלת באופן אוטומאטי על ידי ה-JVM כאשר אובייקט מגיע אל סוף חייו רגע קט לפני ששטח הזיכרון שלו משוחרר. כיוון שכל המחלקות שמגדירים ב-JAVA יורשות באופן אוטומאטי מהמחלקה Object (אם אינך מבין את משמעות ההורשה, כדאי שתחזור ותקרא שורות אלה לאחר קריאת הפרק שדן בהורשה), ניתן לומר שהמתודה finalize() מופעלת באופן אוטומאטי על ידי ה-JVM על כל אובייקט (מכל סוג שהוו) "רגע קט" לפני שהוא מסיים את חייו. במחלקה חדשה שמגדירים ניתן לכתוב את המתודה הזו מחדש (overiding) ובכך לצקת תוכן לפעולתה. התפקיד שמיועד בדרך כלל למתודה finalize() הוא לשחרר משאבי מערכת: לסגור קבצים, לסגור דרכי תקשורת וכדומה, וכמו כן, גם פעולות שיש לבצע, ושהמתכנת עלול לשכוח את ביצוען.

יש מספר כללים ביצירתה של המתודה finalize:

1. יש לתת לה את השם finalize()
2. אסור שיהיו לה פרמטרים
3. עליה להחזיר void
4. עליה להיות עם הרשאת הגישה protected

תנאים אלה נדרשים כדי שהיא תבוא במקום המתודה finalize() אשר מועברת בהורשה מהמחלקה Object.

התפקיד של ה Garbage Collector הוא לנהל מעקב אחר שטחי הזיכרון שמוקצים באופן דינמי עבור האובייקטים בתכנית ולשחרר את אותם שטחי זיכרון של האובייקטים שה-reference אליהם איננו מוחזק באף משתנה (הגיעו לסוף חייהם). ה Garbage Collector גם קורא להפעלת המתודה finalize על האובייקט שהגיע לסוף חייו רגע לפני שהזיכרון שלו משוחרר. כיוון שה-Garbage Collector עצמאי בפעולתו אנו אף פעם לא יכולים להיות בטוחים שה- Garbage

© 2007 Zindell Technologies, Ltd.

Collector אכן הספיק להפעיל את המתודה finalize על האובייקט שסיים את חייו. האפשרות שהתכנית תסתיים לפני שכל זה יתרחש תמיד קיימת. מסיבה זו אין לסמוך על כך שהמתודה finalize אכן תפעל ואין למקם בתוכה פעולות שחשוב שתתבצענה כאשר אובייקט הגיע לסוף חייו.

יש לשים לב להבדלים. לעומת C++: ב-JAVA המתודה finalize לא אחראית לשחרור זיכרון שהוקצה באופן דינמי כמו ה-destructor ב-C++. ההקצאה של זיכרון באופן דינמי ושחרורו נעשים באחריות ה JVM, ולא באחריות המתכנת כמו ב C++.

ניתן להפעיל את finalize באופן יזום, אך יש לזכור כי פעולה זו לא תגרום לשחרור הזיכרון שמשמש לאחסון האובייקט. שחרור הזיכרון מתבצע רק על ידי ה- Garbage Collector.

לסיכום, אם הקדשת מזמנך היקר יותר משתי דקות לקריאת ההסבר על finalize() עליך לעבור לנושא הבא. מעט מאד ספרים מזכירים את קיומה, וזאת כיוון שהיא איננה פונקציה חשובה כפי שהיה ה-destructor ב-C++ .

© 2007 Zindell Technologies, Ltd.

שמירת ההגדרה של המחלקה בתוך קובץ - Class Definitions and Source Files

ב-JAVA אין header files כמו ב-C\C++. כאשר מבוצעת קומפילציה להגדרה של מחלקה, הקומפיילר בודק גם את המחלקות האחרות, שהמחלקה האמורה עושה בהן שימוש (הוא בודק את קבצי ה-class המתאימים). הקומפיילר בודק אם השימוש שנעשה תואם לאופן שבו כל מחלקה ומחלקה מוגדרת.

בכל קובץ קוד מקור לא ניתן לשמור יותר מהגדרה אחת של מחלקה עם הרשאת הגישה public. מספר המחלקות (עם הרשאת גישה ששונה מ-public) שניתן להגדיר בתוך קובץ אחד איננו מוגבל. מחלקה שלא צוינה בהגדרתה הרשאת גישה תיחשב למחלקה שניתן להשתמש בה אך ורק בגבולותיהן של מחלקות ששייכות לאותו package.

אם קובץ קוד מקור מכיל הגדרה של מחלקה עם הרשאת הגישה public אז שמו חייב להיות זהה לשם של המחלקה עם הרשאת הגישה public. הזהות צריכה להיות מושלמת ועם רגישות לאותיות גדולות וקטנות. כך למשל, כיוון ששם המחלקה מתחיל באות גדולה גם שם הקובץ צריך להתחיל באות גדולה. חשוב להקפיד על כך. ה- JVM דורש שכך זה יהיה (שאם לא כן, הוא לא יוכל לפעול כהלכה).

אם הקובץ לא מכיל הגדרה של מחלקה עם הרשאת הגישה public אז אין כל מגבלה בנוגע לשמו.

© 2007 Zindell Technologies, Ltd.

יצירת קבועים - final variables

הדרך ליצירת קבועים בתכנית שכתובה ב-JAVA היא באמצעות המילה השמורה final. יש לרשום final בתחילת שורת ההצהרה של המשתנה שרוצים שייחשב לקבוע, ולאתחל אותו.

ניתן להוסיף את המילה השמורה static בתחילת שורת ההצהרה על הקבוע, ובכך לקבל קבוע שאיננו שייך לאף אובייקט של המחלקה, ויחד עם זאת, כל אחד מהאובייקטים שנוצרו מהמחלקה יכולים לגשת אליו. הוספת static היא הגישה המקובלת ביצירת קבוע.

לדוגמא:

```
public class Car

{

    public final static int MAXSPEED = 100;

    .

    .

    .

}
```

שמות של קבועים כותבים כמו ב-C באותיות גדולות. זוהי המוסכמה שיש לכבד. מבחינה טכנית קבוע יכול להיכתב גם באותיות קטנות, אך זאת יהיה מנוגד למוסכמה. אם המשתנה שיוצרים כקבוע באמצעות final מכיל reference לאובייקט מטיפוס מחלקה כלשהי אז המשמעות להיותו קבוע היא שלא ניתן לשנות את ה-reference שהוא מכיל. את ערכיו של האובייקט, שאותו משתנה מכיל את ה-reference שלו, אין כל מניעה לשנות. הקביעות פועלת על המשתנה בלבד, כלומר על תוכנו (ה-reference) בלבד.

© 2007 Zindell Technologies, Ltd.

יצירת קבוע שאינו static אפשרית, אך היא מאד בזבזנית. יותר זיכרון יצרך. אם הקבוע שמוגדר במחלקה איננו static אז הוא יימצא בכל אחד מהאובייקטים שיווצרו מהמחלקה. זה מאד בזבזני. ב-JAVA אין דרך ליצירת קבוע שדומה לדרך ליצירת קבועים באמצעות #define ב-C. הדרך שהוסברה היא הדרך היחידה.משתנה מסוג final (כלומר, משתנה שהוא קבוע) חייבים לאתחל בשורת הגדרתו. הדוגמה הבאה מדגימה את אופן יצירתו של קבוע ב-JAVA:

```
 class Pon
{
	private int x,y;
	public final static int SIZE=100;
	static int numOfPoints;
	static
	{
		numOfPoints=0;
	}
}
```

הקבוע שמוגדר במחלקה משמש את כל אחד מהאובייקטים שנוצרים ממנה.

© 2007 Zindell Technologies, Ltd.

תכנון נכון של מחלקה Class Design

להלן מספר "כללי אצבע" שמומלץ לפעול על פיהם.

יש להשתדל לקבוע את הרשאת הגישה private לכל משתנה. במתן הרשאת הגישה private למשתנים רבים ככל האפשר מבטיחים יותר את שלמות ונכונות נתוני האובייקט.

יש להשתדל לאתחל משתנים של המחלקה. למרות שמשתנים של מחלקה מאותחלים באופן אוטומטי בערכי ברירת המחדל על פי טיפוס הערך של כל אחד ואחד מהם, איתחול מכוון מקנה בהירות רבה יותר לקוד.

יש להשתדל להמעיט במספרם של משתנים מטיפוס בסיסי בהגדרת המחלקה. מחלקה שיש בה מספר רב מאד של משתנים מטיפוס בסיסי מעלה את השאלה שמא ניתן היה להגדיר במקום מחלקה אחת מספר מחלקות, ובכך להקנות לתכנית רמה גבוהה יותר של מודולריות.

יש לקבוע מתודות שמאפשרות גישה למשתנים הפרטיים של המחלקה רק כאשר יש צורך בכך. משתנים שמופיעים בתוך הגדרתה של המחלקה עם הרשאת הגישה private – כדי לאפשר את הגישה אליהם תוך כדי פעולתן של מתודות ממחלקות אחרות יש לכתוב לפחות שתי מתודות מתאימות. האחת לצורך קבלת ערכו של המשתנה, והשניה לצורך קביעת ערכו של המשתנה. כתיבת שתי מתודות אלה לכל משתנה שיש לו את הרשאת הגישה private גם כשאין בכך צורך פוגעת בבהירות של הקוד, ופוגעת במטרה שיש להרשאת הגישה private.

יש להשתמש בסכימה קבועה ועקבית של הגדרת מחלקה. הקפדה על תבנית ברורה בהגדרתה של מחלקה תורמת לבהירות הקוד. כך למשל, אם מחליטים כי המשתנים של המחלקה יופיעו לפני המתודות שלה, ומקפידים על כך בכל המחלקות שמגדירים בהירות הקוד גדלה. באופן דומה, הקפדה על סגנון נאה, והערות עקביות ומסודרות באמצעות ה-

© 2007 Zindell Technologies, Ltd.

javadoc – כל זה גם תורם לבהירות הקוד.

יש לפצל מחלקות למחלקות קטנות יותר כאשר זאת אפשרי. בדומה למה שכבר נאמר קודם, מספר רב יותר של מחלקות מקנה לתכנית רמה גבוהה יותר של מודולריות. חשוב להשתדל לעבוד בכיוון זה.

יש לתת למחלקות ולמתודות שמוגדרות בתוכן שמות שמעידים על פעולתן. מתן שמות שניתן להבין מהם את תפקיד המחלקה, תפקיד המתודה וגם תפקידו של המשתנה חשוב ביותר. בהירות הקוד עוזרת גם למי שמתכנת וגם למי שרק מסתכל של התכנית הנכתבת להבין טוב יותר את פעולתה של התכנית.

© 2007 Zindell Technologies, Ltd.