

## פרק 2

# עוברים לג'אווה

המטרה של פרק זה ושל המעבדות הנלוות אליו היא לעזור לכם לרכוש יכולת תכנותית בג'אווה, ברמה דומה לזו שיש לכם בשפה הפרוצדורלית שאותה כבר למדתם (פסקל או C), ובנוסף, להכיר את סביבת העבודה JCreator שאתה נעבוד. בפרק זה, שלא כמו בשאר פרקי הספר, לא נעסוק בתכנות מונחה עצמים, אלא נלמד כמה כללי כתיבה בסיסיים בג'אווה, ומגוון פקודות כגון פקודות בקרה ושליטה בזרימת תוכנית. ידע זה יאפשר לכם לכתוב תוכניות בסגנון תכנות פרוצדורלי, תוך שימוש בג'אווה. בפרקים הבאים נתקדם אל לב העניין – עקרונות התכנות מונחה העצמים – ונשתמש בג'אווה ככלי למימוש עקרונות אלו.

טיפוסי הנתונים והפקודות הבסיסיות של ג'אווה דומים מאוד לאלו הקיימים בשפות פרוצדורליות כמו פסקל או C. לכן רוב החומר בפרק זה יהיה מוכר לכם, לפחות באופן חלקי, מלימודיכם בשפות תכנות קודמות.

שימו לב לכל פרט בקטעי הקוד המובאים במהלך הפרק, ונסו להבין מה מבצעת התוכנית בכל שלב. ייתכן כי כעת אינכם מזהים את כל סימני הקוד, אולם הם יוסברו במהלך הפרק ובמעבדות הנלוות אליו, כך שבסיום פרק זה תוכלו להבין את הקוד באופן מספק.

### א. תוכנית ראשונה בג'אווה

#### איך אפשר לקרוא את זה?

עובדה, אפשר! אם כבר למדתם שפת תכנות כלשהי, הרי שכמה מהסימנים והפקודות המופיעים בתוכנית הבאה בוודאי מוכרים לכם. קראו את ההערות המופיעות בתוכנית ולפיהן נסו לפענח מה מבצעת התוכנית.

```
/**
 * This program prints the student's full name.
 */
public class StudentDetails{
    public static void main (String[] args){
        //Prints the student's first name
        System.out.println (" The student's first name is: Albert ");
        //Prints the student's last name
        System.out.println (" The student's last name is: Einstein ");
    } //end of main
} //end of class
```

#### 2.1 תוכנית

על מנת להבין בדיוק מה כתוב בתוכנית 2.1, ולא רק לנחש, נתחיל את צעדינו הראשונים בלימוד שפת ג'אווה.

## הצגת פלט

בג'אווה ניתן להציג פלט טקסטואלי רגיל על המסך (console), על ידי הפעלה של `System.out.print(...)` ו-`System.out.println(...)`. השיטות `print(...)` ו-`println(...)` מקבלות בסוגריים את הפרמטרים שאותם יש להציג. השיטה `println(...)` מבצעת גם מעבר לשורה חדשה לאחר הצגת הפלט.

השורה הבאה לקוחה מתוכנית 2.1:

```
System.out.println (" The student's first name is: Albert ");
```

שורה זו מציגה על המסך את המחרוזת הנתונה בין מרכאות, ובכלל זה הרווחים המופיעים בתחילתה, בסופה ובין התווים. כיוון שבהמשך התוכנית מציגים מחרוזות נוספת בשורה חדשה, יש להשתמש ב-`System.out.println(...)` ולא ב-`System.out.print(...)`. בהמשך נראה כיצד אפשר להציג נתונים מטיפוסים שונים על ידי שימוש בשיטות אלו.

אף על פי שהצגת פלט נעשית בג'אווה באופן פשוט, קבלת קלט אינה פעולה קלה כל כך. אולם, בדפי המעבדה המצורפים לפרק זה נלמד כיצד נוכל בכל זאת לקבל קלט מהמשתמש בצורה נוחה.

## מילים שמורות

המילים השמורות של שפת ג'אווה שבהן השתמשנו בתוכנית 2.1 הן המילים המוטות והמודגשות: `class`, `public`, `static`, `void`. במהלך הלימוד נסביר את המילים הללו, ונלמד כיצד להשתמש בהן. בנספח 2 תוכלו למצוא את כל המילים השמורות הקיימות בג'אווה לפי סדר הא"ב.

## ב. כללים ומוסכמות

לכל סיפור או שיר שאנו קוראים בעברית קיימים סימנים וכללים תחביריים שעוזרים לנו להבינו ולפרשו, כגון חלוקה לבתים ולמשפטים וסימני פיסוק שונים. בג'אווה, כמו בשפות תכנות אחרות שהכרתם, קיימים מספר סימנים תחביריים המאפשרים לנו ולמהדר לקרוא ולהבין את הקוד. ישנם סימנים תחביריים, שאם לא נכתוב אותם במקומות הדרושים, לא נצליח להריץ את התוכנית, והמהדר יציג לנו הודעת שגיאה.

1. נקודה-פסיק ; – סימן זה חייב להופיע בסוף כל משפט תכנותי (הצהרה או פקודה).

הנקודה-פסיק היא חלק בלתי נפרד מכל פקודה והיא מורה על סיומה.

2. סוגריים מסולסלים שמאליים { וסוגריים מסולסלים ימניים } – זוג סימנים זה מורה על

פתיחה וסגירה של בלוק פקודות בהתאמה. הסוגריים המסולסלים תוחמים בלוק של

פעולות אשר מתבצעות כיחידה אחת בזו אחר זו. כמובן שבכל תוכנית צריך לשמור על

איזון סוגריים.

פרט לכללי החובה הללו, יש כללי כתיבה סגנוניים שיעזרו לכם ולאנשים אחרים להבין את הקוד. שמירה עליהם מאפשרת כתיבת קוד נקי ומסודר שאותו ניתן לתחזק בקלות. כללים אלו התקבלו על ידי מתכנתים כמוסכמות של סגנון תכנות בשפת ג'אווה.

1. **הזחה (Indentation):** כאשר אנו כותבים פעולות בתוך בלוק מסוים אנו מזיחים אותן, כלומר מרחיקים אותן מהשוליים מעט ימינה על ידי מקש tab (לעתים יעזור לנו העורך בסביבת העבודה ויעשה זאת אוטומטית). בזכות פעולת ההזחה המבנה הצורני של התוכנית ברור וקריא ותואם את המבנה הלוגי שלה.
  2. שורות התוכנית צריכות להיות מחולקות במקומות נוחים כך שהתוכנית תהיה קריאה. כמו כן, אין לכתוב יותר מפקודה אחת בשורה.
  3. שמות ומשמעותם: בג'אווה נהוג לכתוב שמות בעלי משמעות ברורה וקריאה (גם אם השמות ארוכים). כאשר שם כלשהו (מחלקה, שיטה או משתנה) מורכב מכמה מילים מקובל להתחיל כל מילה, חוץ מהראשונה, באות גדולה. שמות של מחלקות מתחילים באות גדולה, לדוגמה: StudentDetails. שם המחלקה צריך להתאים לתוכן שלה ולהבהיר מה היא מייצגת. בג'אווה, בניגוד לשפות תכנות רבות אחרות, אורך השמות אינו מוגבל.
  4. **הערות (comments):** באמצעות הערות אנו יכולים לשלב בתוכנית הסברים על אודות הקוד. הסבר צריך לאפשר לכל אחד להבין מה הייתה כוונת המתכנת. לכן, כתיבת הערות היא נדבך חשוב מאוד של סגנון תכנותי טוב. בג'אווה קיימות מספר שיטות תיעוד. נציג שתי שיטות: אם נרצה לכתוב הערה בשורה אחת נשתמש בסימן קו-נטוי-כפול (//). כל טקסט שיופיע מסימן זה ועד סוף השורה הוא חלק מהערה ואינו חלק מהקוד. לעומת זאת, אם נרצה לכתוב הערה על פני יותר משורה אחת, נשתמש בסימן קו נטוי וכוכבית (/\*) על מנת לפתוח את ההערה, ובסימן כוכבית וקו נטוי (\*/) על מנת לסגור אותה. המהדר "מבין" כי הטקסט שבין סימנים אלה אינו חלק מקוד התוכנית ומתעלם ממנו.
- תוכנית תרוץ גם אם לא נקפיד על ארבעת הכללים האחרונים שציינו, אולם כאשר אנו כותבים תוכנית, כדאי מאוד לדאוג לכך שהיא תהיה ברורה לא רק לנו ולמחשב, אלא לכל מי שיקרא את הקוד. לכן, לאורך הספר כולו וגם בכיתה, במעבדה ובבחינות(!) נקפיד על כללים אלה לא פחות מאשר על כללי התחביר של השפה.
- מידע נוסף על מוסכמות הכתיבה בשפת ג'אווה תוכלו למצוא באתר חברת סאן, בכתובת:
- <http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>

## ג. שיטות

**שיטה (method)** היא פקודה או סדרה של פקודות בשפת תכנות, המאוגדות תחת שם אחד לצורך ביצוע פעולה מסוימת. כאשר קוראים לשיטה מסוימת במהלך התוכנית, סדרת הפקודות שהיא מכילה, מתבצעת. בסיום הביצוע התוכנית חוזרת למקום שממנו השיטה נקראה, וממשיכה משם. בכל תוכנית ג'אווה תופיע שיטה ששמה main(...), שממנה מתחיל ביצוע התוכנית. כדי לקרוא לשיטה של אובייקט כלשהו נשתמש ב"סימון-הנקודה" (**dot notation**). לדוגמה, כדי לקרוא לשיטה go() של האובייקט robot, נרשום:

```
robot.go();
```

באמצעות פקודה זו אנחנו מבקשים מהאובייקט robot לבצע את השיטה go() המוגדרת אצלו. את שם האובייקט נרשום משמאל לנקודה, ואת שם השיטה שלו מימין לנקודה.

בתוכנית 2.1 השתמשנו בסימון-הנקודה כדי לקרוא לשיטות `print(...)` ו-`println(...)` של האובייקט `out`. במילים אחרות, ביקשנו מהאובייקט `out` להפעיל את השיטות המצוינות מימין לסימון-הנקודה. לסימון-הנקודה יש שימוש נוסף בפקודות ההדפסה שראינו: בין המילה `System` לבין האובייקט `out`. על נקודה זו לא נדון בשלב זה.

---

? כיצד תיראה תוצאת הרצת תוכנית 2.1?

---

## ד. טיפוסים נתונים בסיסיים

גיאווה מעמידה לרשותנו שמונה טיפוסים נתונים בסיסיים (**primitive data types**), וביניהם: מספרים שלמים, מספרים עשרוניים ותווים. כל טיפוס יכול לייצג קבוצת ערכים, וניתן לבצע עליו אוסף של פעולות. כך למשל, הטיפוס הבסיסי `int` יכול לייצג מספרים שלמים. נכיר מקרוב כמה מטיפוסי הנתונים השימושיים ביותר.

### מספרים

בגיאווה מוגדרים ארבעה טיפוסים נתונים בסיסיים עבור מספרים שלמים: `byte`, `short`, `int`, `long` ושני טיפוסים נתונים בסיסיים עבור מספרים עשרוניים: `float`, `double`. ששת הטיפוסים הללו נבדלים זה מזה בכמות הזיכרון המוקצית לערכים שלהם. כתוצאה מכך, כל טיפוס יכול להכיל ערכים בטווח שונה. למשל, עבור הטיפוס הבסיסי `byte` מוקצית כמות הזיכרון הקטנה ביותר למספר שלם: המספר הגדול ביותר שניתן לאחסן במשתנה מטיפוס זה הוא 127. לעומת זאת, עבור הטיפוס `long` מוקצית כמות הזיכרון הגדולה ביותר עבור מספר שלם: המספר הגדול ביותר שניתן לאחסן במשתנה מסוג זה הוא 9,223,372,036,854,775,807.

על פי רוב, כדי לשמור מספרים שלמים נשתמש בטיפוס הבסיסי `int`, אשר מחד גיסא, טווח המספרים השלמים שהוא מייצג יספיק למרבית היישומים שלנו, ומאידך גיסא, הוא אינו תופס זיכרון רב מדי. בעבור מספרים ממשיים נבחר להשתמש בדרך כלל בטיפוס הבסיסי `double`. בנספח 3 תוכלו למצוא טבלה המפרטת מהי כמות הזיכרון המוקצית לכל טיפוס ומהו טווח המספרים שכל טיפוס יכול לייצג. הפעולות שניתן לבצע על טיפוסים נתונים אלה הן הפעולות האריתמטיות המוכרות לכם, כגון חיבור, חיסור, כפל וחילוק.

### תווים

טיפוס בסיסי נוסף המוגדר בשפה נקרא `char`. טיפוס זה כולל את כל התווים האפשריים (אותיות וסימנים) כמעט בכל השפות בעולם. תו מיוצג בגיאווה על ידי תחיתתו בין גרש שמאלי לגרש ימני כך: `'a'` או `'.'`. לעומת זאת, מחרוזות מיוצגות בגיאווה על ידי תחיתתן בין מרכאות כפולות כך: `"a"` (מחרוזת באורך 1) או כך: `"Always look on the bright side of life"`. שימו לב, **מחרוזת (String)** אינה טיפוס בסיסי, אלא אובייקט. נרחיב את הדיבור על מחרוזות בהמשך, בפרק 5 "פענוח צפונות ה-`main(...)`".

## טיפוס נתונים בוליאני

טיפוס נתונים בוליאני מכיל את הערכים הלוגיים: **אמת** (*true*) או **שקר** (*false*). בג'אווה נקרא טיפוס נתונים זה *boolean*. משתנה בוליאני משמש בדרך כלל לבדיקת נכונותם של תנאים. בהמשך נדון בתנאים ובלולאות, ונראה כיצד ביטויים בוליאניים משמשים אותנו לשם ביצוע פעולות אלה.

## משתנים

בזמן ריצת תוכנית יש לשמור נתונים בתאי זיכרון. כדי לעשות זאת, נשתמש במשתנים. טיפוס המשתנה קובע את סוג הערכים שניתן להציב בו, את הפעולות שניתן לבצע עליו ואת נפח הזיכרון המוקצה לו. כדי להצהיר על משתנים נציין את טיפוסו של המשתנה ואחר כך את שמו. נתבונן בדוגמה הבאה:

```
int x;
```

הצהרנו על משתנה בשם *x* מטיפוס *int*. כלומר בקשנו להקצות עבור *x* נפח זיכרון בגודל המתאים לאחסון *int*.

יש לבחור שמות משתנים בעלי משמעות. למשל, כדי לשמור את הציון הממוצע של תלמיד בתרגילים בפיזיקה נוכל להצהיר על המשתנה כך:

```
double averageGradeInPhysics;
```

הצהרנו על משתנה בשם *averageGradeInPhysics* מטיפוס *double*, משום שהציון הממוצע של תלמיד יכול להיות מספר לא שלם, כמו למשל 88.75. נוכל גם להצהיר על משתנה בוליאני, אשר יכיל את הערך *true* אם התלמיד עבר בחינה ואת הערך *false* אם התלמיד לא עבר את הבחינה. נעשה זאת כך:

```
boolean passedTest;
```

שם המשתנה הבוליאני הוא *passedTest*, וכעת לאחר שהצהרנו עליו יוקצה עבורו מקום בזיכרון שיוכל להכיל רק אחד משני הערכים הלוגיים: אמת או שקר.

---

**?** כיצד תיראה הצהרה על משתנה שבו נשמור את האות הראשונה של שם התלמיד?

---

## הצבת ערכים במשתנים

את ההצהרה על המשתנה ניתן לעשות בכל שלב בתוכנית, אולם לפני שמשתמשים בו, חובה עלינו לדאוג לכך שיהיה בו תוכן: עלינו להציב בו ערך המתאים לטיפוס המשתנה. כדי להציב במשתנה *gradeInPhysics* את הערך 88.5 נכתוב:

```
gradeInPhysics = 88.5;
```

בגיאווה ניתן להצהיר על משתנה ולהציב בתוכו ערך באותה הפקודה, או להצהיר תחילה על משתנה ורק לאחר מכן להציב בו ערך. אנו נעדיף לאתחל משתנים ולהציב בהם ערך כלשהו עם הצהרתם.

נתבונן בדוגמה הבאה, ונראה כיצד אנו מצהירים על משתנים ומציבים בהם ערכים שונים:

```
/**
 * This program prints the student's full name
 */
public class StudentDetails{
    public static void main (String[] args){
        //Prints the student's full name
        System.out.print ("The student's first name is: Albert" + "\n");
        System.out.print ("The student's last name is: Einstein" + "\n");
        double gradeInPhysics = 88.5;
        System.out.print ("Grade in Physics:" + gradeInPhysics + "\n");
        double bonus = 11.5;
        gradeInPhysics = gradeInPhysics+bonus;
        System.out.print ("Grade in Physics:" + gradeInPhysics);
    } //end of main
} //end of class
```

## 2.2 תוכנית

האופרטור + שימש בתוכנית בשני אופנים שונים המוכרים לכם בוודאי. מחד גיסא, הוא שימש כסימן פעולה חשבוני:

```
gradeInPhysics = gradeInPhysics+bonus;
```

מאידך גיסא, הוא שימש לביצוע שרשור של מחרוזת:

```
System.out.print ("Grade in Physics:" + gradeInPhysics + "\n");
```

בתוכנית 2.2 השתמשנו בשיטת ההדפסה print, שאינה כוללת מעבר לשורה חדשה בתום ההדפסה. כדי לבצע מעבר לשורה חדשה השתמשנו בסימן "\n". משמעות הסימן היא: התחל שורה חדשה. בדומה לסימן "\n" מוגדרים בגיאווה סימנים נוספים הגורמים לאפקטים מיוחדים בעת ההדפסה. בנספח 4 תמצאו רשימה של כל הסימונים הללו ומשמעותם.

---


? כיצד יראה הפלט המלא של תוכנית 2.2?

---

## המרה בין טיפוסים משתנים

בגיאווה חובה להציב במשתנה ערך המתאים לטיפוס של משתנה זה. למשל, אם משתנה מוצהר כמשתנה בוליאני, תהיה זו טעות לנסות ולהציב בו ערך מספרי. עלינו לשים לב שהביטוי בצד הימני של שורת ההצבה הוא מאותו טיפוס של המשתנה הנמצא בצד השמאלי.

לדוגמה, ההצבה הבאה אינה חוקית:

 `int price = 9.99;`      `int //` שגיאה לוגית! הצבת מספר עשרוני במשתנה מטיפוס

הצבה של נתון מטיפוס מסוים במשתנה מטיפוס אחר היא שגיאה לוגית, ולרוב היא תוביל להכרזה על טעות על ידי המהדר.

קיימים מקרים שבהם ההמרה בין טיפוסים נתונים תתבצע באופן אוטומטי אף על פי שהנתון והמשתנה אינם מאותו טיפוס. למשל, אם משתנה מוגדר כ-`double`, ניתן להציב בו גם מספר שלם (`int`). את הערך שנציב ג'אוה תהפוך אוטומטית ממספר שלם למספר עשרוני. אם נכתוב:

```
double x = 7;
```

ג'אוה תמיר אוטומטית את המספר 7 למספר העשרוני 7.0 ורק אז המספר יוצב במשתנה `x`. על מנת שיהיה ניתן להמיר את טיפוס הנתונים יש להקפיד ששום מידע לא יאבד כתוצאה מההמרה. כך למשל, המרה של מספר עשרוני למספר שלם עלולה לגרום לאובדן החלק שמימין לנקודה העשרונית, ולכן המהדר לא יבצע אותה באופן אוטומטי, אלא יציג הודעת שגיאה. אם נרצה לבצע את ההמרה באופן חוקי, נכתוב זאת במפורש על ידי שימוש בפעולת **עיצוב (casting)**. נעשה זאת כך:

```
int price = (int) 9.99; // בצד הימני של השוויון מתבצעת המרת ערך עשרוני לערך שלם
```

החלק העשרוני של המספר יושמט, המספר העשרוני 9.99 יומר למספר השלם 9 ואז יוצב הערך השלם במשתנה `price`. פעולת העיצוב גורמת לאובדן מידע של הערך המקורי. לאחר שהחלק העשרוני של המספר הושמט, ג'אוה מתייחסת אל הביטוי בצד ימין כאילו היה מטיפוס `int`.

## קבועים

קבוע הוא ערך הנשמר לאורך כל ריצת התוכנית, ואינו ניתן לשינוי. ההצהרה על קבוע דומה מאוד להצהרה על משתנה, אלא שלהצהרה על קבוע מתוספת המילה השמורה `final` שמבהירה למהדר כי מדובר בקבוע. בקבוע מותר להציב ערך פעם אחת בלבד. הצבה זו קובעת את ערכו הסופי ואין אפשרות לשנות ערך זה בהמשך התוכנית. בג'אוה מקובל לכתוב שמות של קבועים באותיות גדולות. המילים המרכיבות את שם הקבוע מופרדות בעזרת קו תחתי (`_`). נגדיר ציון מקסימלי כקבוע:

```
final int MAX_GRADE = 100;
```

הצהרנו על קבוע מסוג מספר שלם ששמו הוא `MAX_GRADE`, והצבנו בו את הערך 100. בכל פעם שנרשום את שם הקבוע בתוכנית הדבר יהיה שקול לכתובת המספר 100. שימוש בקבועים הוא הרגל תכנותי טוב למקרים שבהם ידוע שערך מסוים לא ישתנה במהלך ריצת התוכנית. הרגל זה מונע שינויים בקוד שעלולים להיעשות בהיסח הדעת, והופך את הקוד

קריא יותר וקל לתחזוקה. אם נרצה לשנות את הציון המקסימלי ל-10 במקום 100, אזי אין צורך לחפש את כל המקומות שבהם מופיע המספר 100 בתוכנית ולשנותם, אלא נוכל לשנות פעם אחת את הגדרת ערך הקבוע MAX\_GRADE מ-100 ל-10.

---

**? כיצד תשמרו את המספר פאי ( $\pi$ ) כקבוע?**

---

## ה. תנאים ולולאות: שליטה בזרימת תוכנית

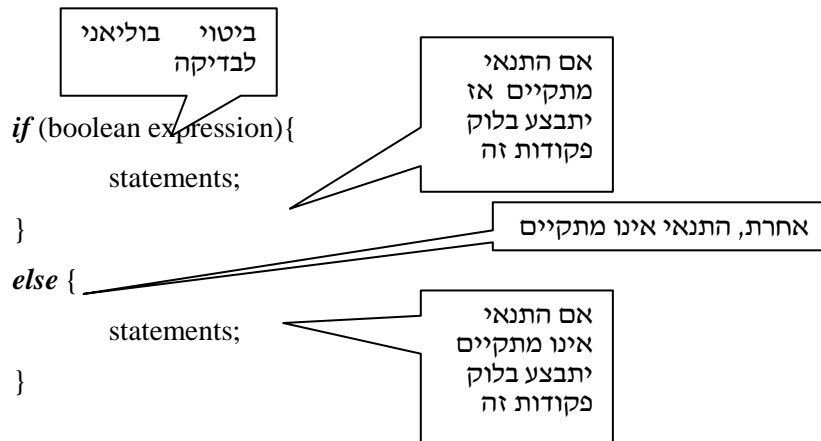
כל תוכנית ג'אוה מתבצעת מהפקודה הראשונה של השיטה main(...) ועד הפקודה האחרונה שמופיעה בה. כמו בשפות התכנות המוכרות לכם, גם בג'אוה קיימות "פקודות לשליטה בזרימת תוכנית" (program flow control operations), המאפשרות לנו לשנות את הביצוע הסדרתי של התוכנית על ידי בחירה בין אפשרויות או על ידי ביצוע חזרות.

### התנאי *if-else*

צמד המילים השמורות *if-else* משמש לבניית פקודת תנאי שימושית ביותר לשליטה בזרימת תוכנית. נשתמש בפקודה *if* על מנת לבחור בין אפשרויות על פי ערכו של תנאי לוגי הנבדק בזמן ריצת התוכנית. אם התנאי שנבדוק מתקיים, אזי תתבצע סדרת הפקודות המופיעה בבלוק שאחריו. אחרת, במקרה שהתנאי אינו מתקיים, התוכנית תדלג אל המילה *else* ותבצע את סדרת הפקודות המופיעה בבלוק שלאחריה.



באופן כללי פקודת התנאי *if-else* נראית כך :



המבנה מכיל שני "ענפים": "ענף" *if*, ואחריו "ענף" *else*. גם מבנה תנאי שבו קיים "ענף" אחד, כלומר מופיע *if* בלבד, הוא חוקי.

התנאי שבסוגריים העגולים לאחר פקודת ה-*if* הוא ביטוי בוליאני. בדרך כלל ניצור ביטויים מסוג זה תוך שימוש בסימני יחס (<, >, <=, >=, ==, !=). בנספח 5 תוכלו למצוא טבלה שבה מפורטים סימני יחס אלו ומשמעותם.

### == אינו שווה ל =

סימן השווה הכפול (==) משמש בג'אווה להשוואה לוגית. לעומת זאת, סימן השווה הבודד (=) משמש להצבת נתונים במשתנים, כפי שראינו בדוגמאות הקודמות. הדמיון הצורני בין שני סימנים אלו עלול להיות מקור לטעויות רבות. נתבונן בתוכנית הבאה :

```
/** This program prints different messages using if-else.*/
```

```
public class CheckJavaProgrammer{
    public static void main (String[] args){
        int javaProgrammer;
        ... // javaProgrammer gets its value here
        if (javaProgrammer == 0) {
            System.out.println ("Go study!");
            System.out.println ("Java is an important language");
        }
        else {
            System.out.println ("Great!");
            System.out.println ("Can you start working today? ");
        }
    } //end of main
} //end of class
```

### תוכנית 2.3

התוכנית אמורה להציג פלט בהתאם לערכו של המשתנה `javaProgrammer`. אם ערך הביטוי הבוליאני בתנאי הוא `true`, כלומר ערך המשתנה `javaProgrammer` הוא 0, תבצע התוכנית את שתי פקודות ההדפסה שמופיעות בבלוק הראשון.

אחרת (אם ערך הביטוי הבוליאני הוא `false`), התוכנית תבצע את פקודות ההדפסה הנמצאות בבלוק השני המופיע לאחר המילה השמורה `else`. אפשר לקנן מספר פקודות `if-else` זו בתוך זו על מנת לבדוק יותר משתי אפשרויות.

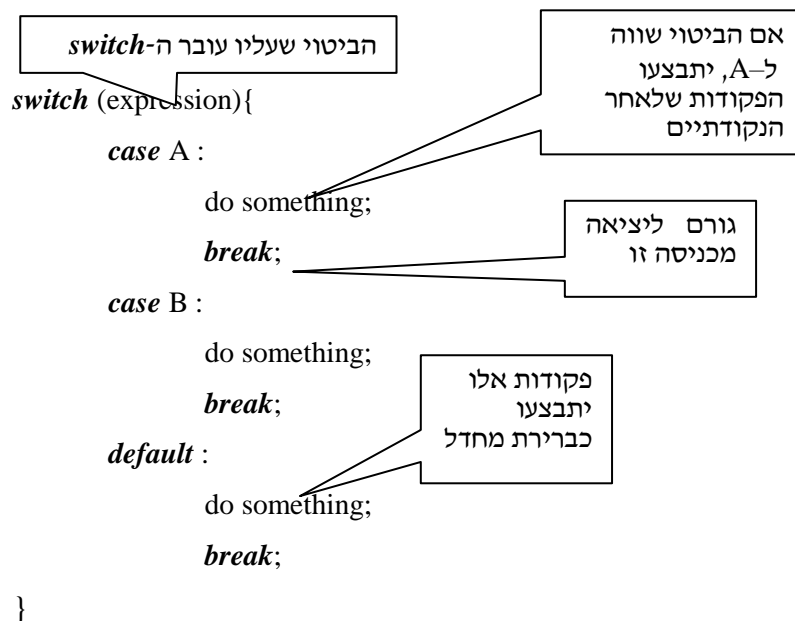
באופן כללי, קינון של מספר פקודות `if-else` יראה כך (או מורכב עוד יותר):

```
if (boolean expression){
    statements;
    if (boolean expression){
        statements;
    }
    else{
        statements;
    }
}
else {
    if (boolean expression) {
        statements;
    }
    else{
        statements;
    }
}
```

## תנאי מרובה: פקודת *switch*

ראינו, שעל מנת שהתוכנית תבצע בחירה בין יותר משתי אפשרויות, יש לכתוב שרשרת ארוכה ומקוננת של תנאי *if-else*. ניתן לעשות זאת גם בדרך אלגנטית וקריאה יותר, על ידי שימוש בפקודה *switch*. הביטוי שעליו מתבצעת פעולת ה-*switch* צריך להיות מטיפוס מספר שלם (למשל *int*). הפקודה תבדוק את ערכו של הביטוי, ובהתאם לכך תבצע את הפעולה הרצויה כפי שהמתכנת הגדיר.

באופן כללי פקודת התנאי *switch* נראית כך:



בסוגריים העגולים יכתב הביטוי שעליו תתבצע הפקודה. לאחר מכן, באמצעות סוגריים מסולסלים, יפתח בלוק התוחם את כל חלקי פעולת ה-*switch*. המילה השמורה *case* מציינת כל אחד מהמקרים שבהם נרצה לפעול. אם הביטוי יקבל ערך שאינו תואם לאף אחד מהערכים שבהם רצינו לפעול, תתבצע פעולת ברירת המחדל שמופיעה לאחר המילה השמורה *default*. הפקודה *break* המופיעה בסוף רצף פקודות של כל כניסה מוציאה את התוכנית מהפקודה *switch* והתוכנית ממשיכה בביצוע סדרתי של הפקודות.

נתבונן בקטע הקוד הבא:

```
...
final int BEGINNER = 0;
final int BASIC = 1;
final int INTERMEDIATE = 2;
final int ADVANCED = 3;
final int EXPERT = 4;
final int SUPER_EXPERT = 5;
int yourLevelInJava = 0;
...
```

```
//here we read an integer and assign it to yourLevelInJava.
...
switch (yourLevelInJava){
    case BEGINNER: System.out.println ("Go study!");
        break;
    case BASIC: System.out.println ("Well, this is the beginning.");
        break;
    case INTERMEDIATE: System.out.println ("Keep studying.");
        break;
    ...
    case SUPER_EXPERT: System.out.println ("Can you start working today?");
        break;
    default: System.out.println ("Illegal value.");
}
// end of switch
```

#### תוכנית 2.4

במשתנה `yourLevelInJava` מוצב ערך המייצג את רמת השליטה בשפת ג'אווה (ערך של מספר שלם בין 1 ל-5 שהמשתמש מקליד). עבור כל ערך תתבצע הכניסה (*case*) המתאימה. לדוגמה, אם ערך המשתנה `yourLevelInJava` הוא 2 תתבצע הכניסה `INTERMEDIATE` שערכה 2. ההודעה שתוצג תהיה:

Keep studying.

אם ערך המשתנה אינו ערך בין 0 ל-5, תתבצע פעולת ברירת המחדל המופיעה לאחר המילה השמורה *default*, וכפי שניתן לראות, התוכנית תדפיס הודעה המציינת כי הערך אינו חוקי. כאשר לא נעשה שימוש בפקודה *break* בתוך רצף פקודות של כניסה, יתבצעו גם שאר הכניסות שלאחר הכניסה המתאימה עד סיום הפקודה או עד אשר תופיע הפקודה *break*.

למשל, אם נרצה להציג פלט זהה עבור רמות שליטה שונות בג'אווה, נוכל לעשות זאת כך:

```
switch (yourLevelInJava){
    case BEGINNER: System.out.println ("Very Low Level");
        break;
    case BASIC:
    case INTERMEDIATE: System.out.println ("Average Level");
        break;
    case ADVANCED:
    case EXPERT: System.out.println ("High Level");
```

```

        break;
    case SUPER_EXPERT: System.out.println ("Very High Level");
        break;
    default: System.out.println ("Illegal value.");
} // end of switch

```

## תוכנית 2.5

תוכנית 2.5 תריץ את כל הפקודות המופיעות בבלוק ה-*switch*, החל בשורת ה-*case* המתאימה לערך הביטוי הנבדק, ועד שהיא תתקל בפקודת *break*, או עד שתגיע לסוף ה-*switch*.

---

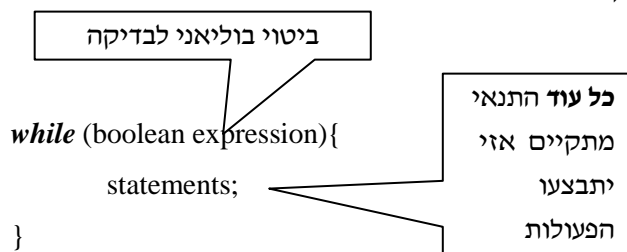
**?** מה תבצע תוכנית 2.5 אם ערך המשתנה יהיה 6? מה היא תבצע אם ערך המשתנה יהיה 0?

---

הפקודות *if-else* ו-*switch* מאפשרות לשלוט בזרימת התוכנית באמצעות בחירה בין מספר אפשרויות. אולם, פקודות אלה אינן מאפשרות לנו לגרום לתוכנית לבצע מספר פעמים אותה פעולה. כדי לגרום לביצוע חוזר של פעולות נשתמש בלולאות.

## לולאות *while* ו-*do-while*

באמצעות פקודת *while* או יכולים לגרום לתוכנית לבצע פקודה או סדרת פקודות פעמים אחדות. כל עוד התנאי שבסוגריים מתקיים, יתבצעו הפקודות הנמצאות בבלוק. באופן כללי המבנה של פקודת *while* נראה כך :



הפקודות המופיעות בבלוק שלאחר הפקודה *while* יתבצעו כל עוד ערכו של התנאי הבוליאני הוא *true*. בכל פעם שהתוכנית תגיע אל הסוגריים המסולסלים החותמים את בלוק הפקודה, היא תשוב ותבדוק אם התנאי מתקיים. אם התנאי עדיין מתקיים, הפקודות שבבלוק יתבצעו שוב. אם התנאי אינו מתקיים, התוכנית תצא מהלולאה ותמשיך בריצתה הסדרתית. אם התנאי אינו מתקיים בפעם הראשונה שהוא נבדק, הפקודות שבבלוק לא יתבצעו אפילו פעם אחת, והתוכנית תמשיך אל הפקודה הבאה שאחרי פקודת ה-*while*.

חשוב לזכור לבצע עדכון נכון בתוך גוף ה-*while*, כך שבשלב כלשהו תנאי הכניסה ללולאה יקבל ערך *false* והלולאה תיעצר. הדבר נכון בדרך-כלל, כאשר אנו אמנם רוצים שהלולאה תיעצר. אולם, ישנם מקרים שבהם אנו מעוניינים בלולאה אינסופית (לדוגמה, חשבו על כספומט שצריך להישאר "חיי" כל הזמן, ולחכות לקלט ממשתמשים חדשים). במקרים כאלו נשתמש בלולאה במבנה הבא :

```

while (true){

```

```

do something
}

```

כעת נשוב לענייננו, לולאות שיש להן התחלה ויש להן סוף.

התבוננו בתוכנית הבאה המציגה טבלת המרה של ערכים בסנטימטרים לערכים באינצ'ים:

```

/** This program converts centimeters to inches.*/
public class ConvertCentimeterToInch{
    public static void main (String[] args){
        final double LOW_VALUE = 2.0;
        final double HIGH_VALUE = 10.0;
        double inch = 0.0;
        double centimeter = 0.0;
        // The following commands prints a table of centimeter converted to inches.
        System.out.println ("CENTIMETERS \t INCHES");
        centimeter = LOW_VALUE;
        while (centimeter <= HIGH_VALUE){
            inch = centimeter / 2.54;    //converts centimeters to inches
            System.out.println (centimeter + "\t\t" + inch);
            centimeter = centimeter + 1.0;
        }//end of while
    } //end of main
} //end of class

```

## 2.6 תוכנית

כל עוד הערך במשתנה `centimeter` קטן מהערך הגבוה שנקבע (`HIGH_VALUE`) או שווה לו, ערך התנאי הבוליאני של הפקודה `while` הוא `true`, ובלוק הפקודות שלאחריו יתבצע. בבלוק זה מתבצעות כמה פעולות: ראשית, הערך שמייצג סנטימטרים מומר לערך השקול לו באינצ'ים. שנית, מוצגים שני הערכים, ושלישית, מתעדכן המשתנה הנבדק בלולאת ה-`while`. הסימן המיוחד `"\t"`, המופיע בסוגריים של פקודת ההדפסה, יוצר רווח של טאב (`tab`) בין ההדפסות.

על מנת להציג טבלה כזו, אפשר היה גם להשתמש בפקודת לולאה `do-while`. בפקודה זו, בניגוד לפקודת `while`, הפעולות המופיעות בבלוק מתבצעות פעם אחת, ורק לאחר מכן נבדק התנאי.

באופן כללי המבנה של פקודת `do-while` נראה כך:

בצע את בלוק הפקודות

```
do{
    statements;
}while (boolean expression);
```

במקרה של שימוש ב- *do-while* התוכנית תיראה כך :

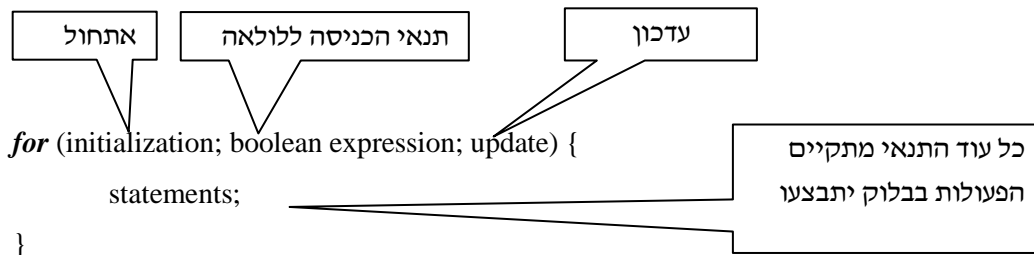
```
/** This program converts centimeters to inches.*/
public class ConvertCentimeterToInch{
    public static void main (String[] args){
        final double LOW_VALUE = 2.0;
        final double HIGH_VALUE = 10.0;
        double inch = 0.0;
        double centimeter = 0.0;

        // The following commands prints a table of centimeter converted to inches.
        System.out.println("CENTIMETERS \t INCHES");
        centimeter = LOW_VALUE;
        do{
            inch = centimeter / 2.54;           //converts centimeters to inches
            System.out.println (centimeter + "\t\t" + inch);
            centimeter = centimeter + 1.0;
        } while (centimeter <= HIGH_VALUE);
    } //end of main
} //end of class
```

## תוכנית 2.7

## לולאת for

פקודת *for* היא דרך נוחה לכתיבת לולאות בג'אווה. לולאה זו דומה ללולאת ה-*while* שכבר הכרנו, אלא שצורת כתיבתה מאפשרת שליטה ברורה יותר במספר החזרות שיתבצעו. בדומה ללולאת *while*, גם בפקודת *for* יש לאתחל את המשתנה בכל פעם מחדש לפני הכניסה אל הלולאה, שכן תנאי הכניסה נבדק לפני כל כניסה מחודשת אל הלולאה, ובסופה הוא מעודכן. גם כאן תנאי הכניסה ללולאה קובע מתי יש לבצע את הקוד שבלולאה: כל עוד ערכו של תנאי זה הוא אמת (*true*), בלוק הלולאה יתבצע. אם ערכו של התנאי הוא שקר (*false*), התוכנית תמשיך בביצוע הפקודות שאחרי סיום הלולאה. נביט תחילה במבנה הכללי של לולאת *for*:



לולאת *for* מורכבת משלושה חלקים המופיעים בסוגריים העגולים הקרויים גם "ראש הלולאה". כל חלק מופרד מהאחר בנקודה-פסיק. סדר הופעת מרכיבי "ראש הלולאה" הוא: אתחול משתנה; תנאי הכניסה ללולאה; עדכון המשתנה.

עדכון המשתנה באופן נכון הוא הכרחי. עדכון שאינו נכון או שאינו מדויק עלול לגרום לכך שהלולאה לא תבצע את מה שהיא אמורה. לדוגמה, אם אנו רוצים להדפיס כל תא זוגי במערך, ועדכון המשתנה נעשה באמצעות האופרטור ++, יהיו הדפסות רבות מדי (פי 2 מהרצוי). העדכון הדרוש במקרה זה הוא  $i = i + 2$ . דוגמה נוספת לתוצאה גרועה של עדכון שאינו נכון היא כניסה ללולאה אינסופית בלתי רצויה. בכל מקרה שבו איננו מעוניינים בלולאה אינסופית חובה עלינו לעדכן את משתנה הלולאה, כך שבשלב כלשהו התנאי יקבל ערך *false* וביצוע הלולאה יעצר.

בלוק הפקודות המופיע אחרי "ראש הלולאה" קרוי "גוף הלולאה". אם ערכו של תנאי הכניסה הוא *true*, הפקודות בגוף הלולאה יתבצעו. בכל פעם שמסתיים ביצוע הפקודות, מעודכן המשתנה המופיע ב"ראש הלולאה", ורק אז נבדק תנאי הכניסה פעם נוספת.

באופן כללי מהלך הביצוע של לולאת *for* נראה כך:

א. אתחול משתנה

ב. בדיקת תנאי

ג. אם התנאי נכון:

1. ביצוע גוף הלולאה

2. עדכון המשתנה שאותחל בשלב א

3. חזרה לשלב ב

אם התנאי הנבדק בשלב ב אינו נכון, נמשיך בביצוע הפקודות שאחרי לולאת ה-*for*. נתבונן בקטע הקוד הבא:



```
for (int i = 10; i >= 0; i--) {  
    System.out.println ("count-down: " + i);  
}
```

## תוכנית 2.8

אנו מעוניינים להדפיס ספירה לאחור מ-10 עד 0. לכן, בראש הלולאה הצהרנו על המשתנה  $i$  ואתחלנו אותו למספר 10. תנאי הכניסה שלנו קובע שכל עוד ערכו של  $i$  גדול מאפס או שווה לו, גוף הלולאה יתבצע, ותתבצע פעולת העדכון שהיא הקטנת  $i$  ב-1. כלומר, גוף הלולאה יתבצע 11 פעמים, כדי לספור מ-10 עד 0 (ועד בכלל). עדכנו את המשתנה על ידי שימוש בפעולה  $i--$  שפירושה הקטנת ערך המשתנה  $i$  ב-1. זוהי צורת כתיבה מקוצרת לפעולה  $i = i - 1$ . בגיאוה קיימים קיצורים נוספים השימושיים לפעולות חשבוניות. קיצורים אלה יופיעו במהלך הדוגמאות שנביא בספר. רשימה של קיצורים שכיחים תוכלו למצוא בנספח 6.

בגוף הלולאה של תוכנית 2.8 מופיעה פקודת הדפסה הכוללת את המשתנה  $i$ . משתנה זה מתעדכן בכל פעם שנכנסים אל הלולאה. לכן, בכל פעם שנבצע את הפקודה יודפס ערכו המעודכן של המשתנה. תוצאת ביצוע הפקודה תראה כך:

```
count-down: 10  
count-down: 9  
count-down: 8  
...  
count-down: 0
```

## ו. סיכום

בפרק זה למדנו לקרוא ולכתוב תוכניות ג'אווה בסיסיות בדומה לאופן שעשיתם זאת בשפות הפרוצדורליות. מטרתנו הייתה להציג את הדקדוק המיוחד של ג'אווה ומספר כללים לתיבת תוכניות בשפה. למדנו מספר כללי כתיבה בג'אווה שיסייעו לנו לכתוב קוד ברור וקריא, למדנו לשמור נתונים במשתנים, לבצע עליהם פעולות וחישובים ולהדפיס אותם. כמו כן, למדנו כיצד לשלוט בזרימת תוכנית ג'אווה על ידי פקודות בקרה.

פרק זה לא עסק בתכנות מונחה עצמים. בפרקים הבאים נתמקד בעקרונות של גישת התכנות מונחה העצמים, ונראה כיצד ניתן לממש באמצעות שפת ג'אווה.

מומלץ לא להמשיך אל הפרקים הבאים בטרם תתרגלו את החומר בעזרת דפי העבודה המצורפים לפרק זה. דפי עבודה אלו יאפשרו לכם להכיר לראשונה את סביבת העבודה שבה נשתמש.

### כללים שיש לזכור

1. נקודה-פסיק בסוף כל פקודה.
2. איזון סוגריים.
3. אינדנטציה, כלומר, קוד מוזח נכון: מחולק באופן לוגי ומסודר.
4. שמות ברורים ובעלי משמעות.
5. שם מחלקה – מתחיל באות גדולה.
6. שם משתנה – מתחיל באות קטנה וכל מילה נוספת בשם מתחילה באות גדולה ללא רווחים בין מילה למילה.
7. שם קבוע – כולו באותיות גדולות עם קו תחתי המפריד בין המילים.
8. הסימן + משמש כפעולת חיבור בין מספרים.
9. הסימן + משמש גם כפעולת שרשור בין מחרוזות.
10. הסימן = משמש כפעולת השמה במשתנה.
11. הסימן == משמש כפעולת השוואה בין ערכים של ביטויים.
12. לכל *else* צריך להיות *if* מתאים שאליו הוא מקושר.
13. פקודת *switch* אחת עדיפה בדרך כלל על פקודת *if-else* שבתוכה שרשרת פקודות *if-else*.
14. ללא *break*, כל המקרים בפקודת *switch* יתבצעו בזה אחר זה.
15. היזהרו מלולאות אינסופיות. יש לבדוק שתנאי הכניסה של לולאה יקבל ערך שקר לאחר מספר סופי של חזרות על הלולאה, אלא אם כן אנו מעוניינים בלולאה אינסופית.

## עיקרי התחביר של שפת ג'אווה

### חתימת השיטה הראשית main(...)

```
public static void main (String[] args){  
    //The program  
}
```

### סימני תחביר בג'אווה

;	סיום הצהרה או פקודה
{...}	פתיחה וסגירה של בלוק פקודות (בשיטה, במבנה בקרה, וכדומה)
//	פתיחת הערה (בת שורה אחת בלבד)
/*...*/	פתיחה וסגירה של הערה (חלק משורה, שורה אחת או יותר)

### טיפוסי נתונים בסיסיים (Primitive Data Types)

<i>int, byte, short, long</i>	מספר שלם כגון 5, -9789
<i>double, float</i>	מספר עשרוני כגון 5.8, -98.00
<i>char</i>	תו בודד (אותיות וסימנים) כגון 'a', '#', '5'
<i>boolean</i>	ערך אמת: <i>true</i> או <i>false</i>

### הצגה סטנדרטית של פלט בג'אווה (Standard Output)

System.out.print ("some string");	הצגת המחרוזת "some string" על המסך
System.out.println ("some string");	זהה לפקודה הקודמת, בתוספת מעבר לשורה חדשה
<i>int</i> k = 5; System.out.println (k);	הצגת תוכן המשתנה k על המסך

### תנאים לוגיים

&&	AND
	OR

## דוגמאות לשימוש בסיסי במשתנים

הצהרה על משתנים

```
int myAge;  
double price;  
float x;  
boolean isRaining;
```

הצבת ערכים במשתנים (לדוגמה באתחול שלהם)

```
myAge = 54;  
x = myAge;
```

הצהרה והצבה (אתחול) בפקודה אחת

```
int myAge = 54;  
int x = myAge;
```

המרה (casting) מטיפוס לטיפוס

```
int basicPrice = (int) 29.99;  
int basicPrice = (int) price;
```

קבועים

```
final double PI = 3.14;
```

## מבני הבקרה שנסקרו במהלך הפרק

<pre><b>if</b> (boolean expression){     statements; } <b>else</b> {     statements; }</pre>	תנאי <i>if-else</i> :
<pre><b>if</b> (boolean expression){     statements;     <b>if</b> (boolean expression){         statements;     }     <b>else</b> {         statements;     } } <b>else</b> {     <b>if</b> (boolean expression) {         statements;     }     <b>else</b> {         statements;     } }</pre>	תנאי <i>if-else</i> מקוננים :
<pre><b>switch</b> (expression){     <b>case</b> A :         do something;         <b>break</b>;     <b>case</b> B :         do something;         <b>break</b>;     <b>default</b>:         do something;         <b>break</b>; }</pre>	תנאי <i>switch</i> :
<pre><b>while</b> (boolean expression){     statements; }</pre>	לולאת <i>while</i> :
<pre><b>do</b>{     statements; }<b>while</b> (boolean expression);</pre>	לולאת <i>do-while</i> :
<pre><b>for</b> (initialization; boolean expression; update) {     statements; }</pre>	לולאת <i>for</i> :

## פרק 2 דף עבודה מס' 1

# סביבת הפיתוח JCreator LE

### מטרות

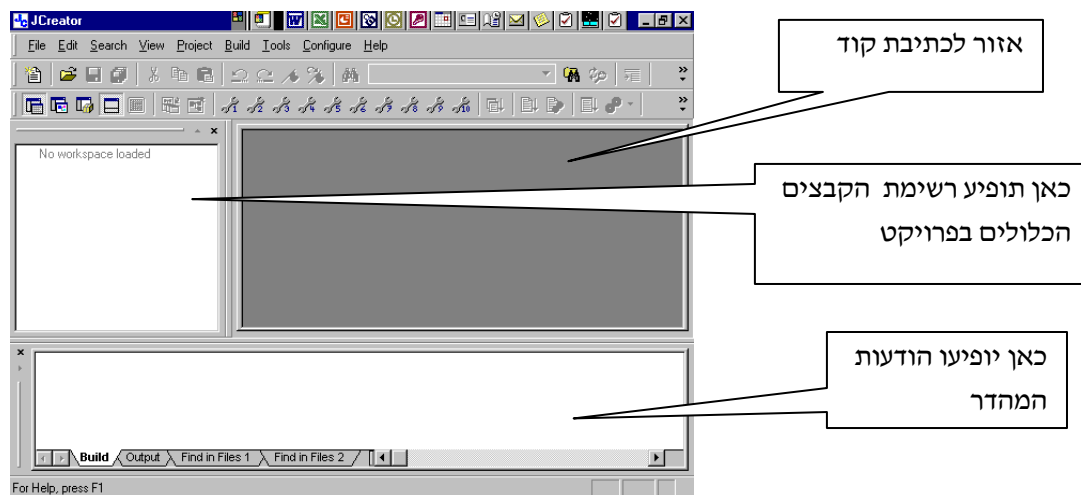
1. צעדים ראשוניים בעבודה עם סביבת הפיתוח JCreator (פתיחת פרויקט, הרצת תוכנית ועוד). מדריך טכני מלא מופיע בסוף הספר.
2. כתיבת תוכנית ראשונה בג'אווה.

### היכרות ראשונית עם סביבת הפיתוח JCreator LE

סביבת הפיתוח שבה נכתוב ונריץ את התוכנית הראשונה שלנו נקראת JCreator LE. בסביבת פיתוח זו נוכל לכתוב תוכניות ג'אווה על ידי שימוש בעורך (editor), ודרכה נוכל גם להדר ולהריץ את התוכניות. כדי להריץ תוכנית Java בסביבת הפיתוח של JCreator ניצור פרויקט. פרויקט הוא דרך להתייחס למספר קבצים **כאוסף**, כלומר להדר אותם יחד, להריץ את התוכנית הראשית המתייחסת אליהם וכדומה.

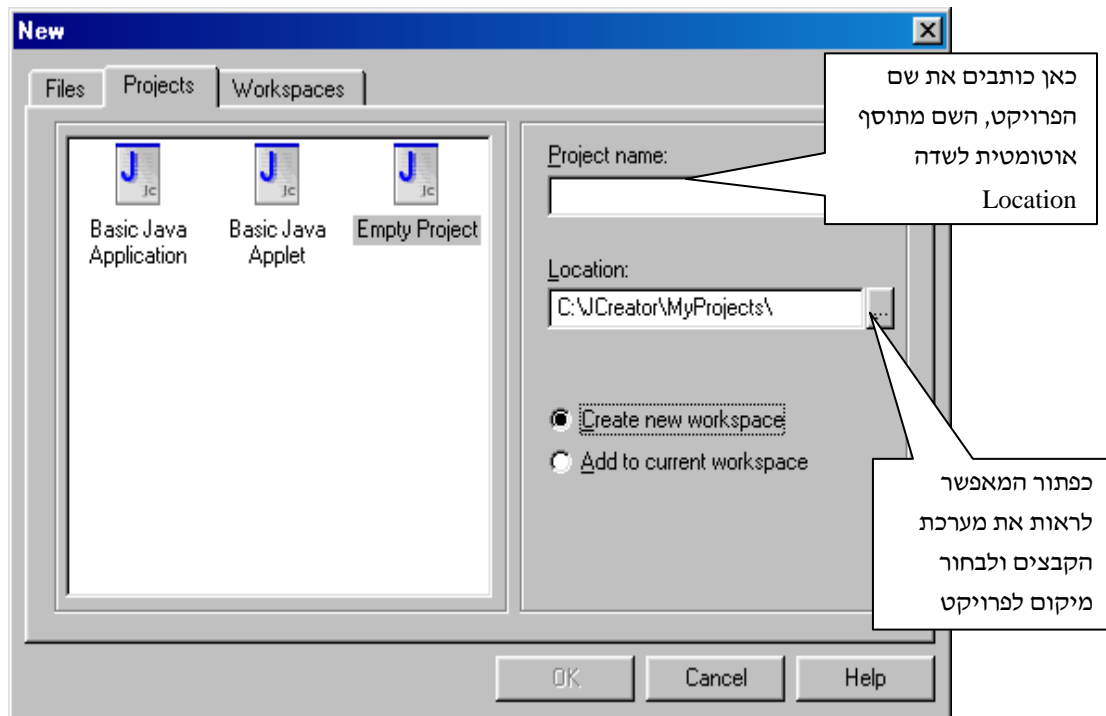
### הרצת סביבת העבודה

כדי להריץ את סביבת הפיתוח עליכם ללחוץ על האייקון של תוכנת JCreator (ראו איור 1), או לבחור ב-JCreator LE > JCreator LE > Programs > Start (אם ברשותכם מערכת הפעלה בעברית: התחל < תוכניות < JCreator LE < JCreator LE). כעת, תוכלו לראות את סביבת הפיתוח (איור 2).



## יצירת פרויקט חדש

1. מתפריט **Project** בחרו ב- **New Project**.
2. נפתחת תיבת דו-שיח, ולה הכותרת **New**.
3. בחרו באייקון **Empty Project**.
4. וודאו כי בשדה **Location** מופיע המסלול אל התיקייה המתאימה לפי הפרק והמעבדה.  
ברירת המחדל לשמירת פרויקטים הקיימת בסביבת הפיתוח היא התיקייה MyProjects (C:\JCreator\MyProjects) מייצג את שם הכוון הקשיח במחשב שעליו התקנתם את JCreator). את הפרויקט הנוכחי כדאי ליצור בתוך C:\JCreator\MyProjects\Chap2 כיוון שהמעבדה שייכת לפרק 2.  
על מנת לבחור מיקום שונה מזה שמופיע, לחצו על הריבוע שמימין לשדה **Location**. יפתח חלון המציג את מערכת הקבצים והמאפשר לבחור מיקום כלשהו (על מנת לבחור מיקום סמנו אותו במערכת הקבצים בעזרת העכבר). סמנו את C:\JCreator\MyProjects\Chap2, לאחר הסימון יופיע המיקום בשדה **Location**.



5. לאחר שבדקתם שאתם יוצרים את הפרויקט בתיקייה הנכונה, עמדו עם הסמן בשדה **Project name** וורשמו את שם הפרויקט שברצונכם ליצור. הפרויקט הראשון שלכם יקרא FullName.
6. סביבת הפיתוח יוצרת באופן מידי **תיקייה** ששמה כשם הפרויקט שלכם. בתיקייה זו עליכם לשמור את כל הקבצים הקשורים לפרויקט. שם הפרויקט החדש מתוסף אוטומטית אל המסלול המופיע בשדה **Location**, כלומר תיקיית הפרויקט שלנו היא לא זאת שסימנתם ב-**Location** אלא תת-תיקייה שלה: <Location>\FullName.
7. סמנו את האפשרות: **Create new workspace**.
8. לחצו על **O.K.**

עד כה למדנו כיצד ליצור פרויקט חדש. כעת נראה כיצד נוכל לכתוב תוכנית ולשמור אותה כחלק מהפרויקט שיצרנו זה עתה.

## כתיבת תוכנית

1. מהתפריט **Project** בחרו ב- **New Class**.
  2. נפתחת תיבת דו-שיח שכותרתה **Class Wizard**, ובה האייקון **Class** (בצד שמאל למעלה) מסומן באופן אוטומטי.
  3. בשדה **Class name** רשמו את שם המחלקה שברצונכם ליצור (שם המחלקה הראשונה שתיצרו יהיה **MyName**).
  4. לחצו על **O.K.**
- הקובץ שיצרתם זה עתה מתוסף באופן מידי אל הפרויקט, וממוקם בתיקיית הפרויקט (התיקייה ששמה כשם הפרויקט שיצרתם). כדי לכתוב את קוד התוכנית הקליקו פעמיים על שם הפרויקט המופיע בחלון השמאלי (**FullName**) ולאחר מכן הקליקו פעמיים על שם הקובץ שאותו אתם מעוניינים לראות (**MyName.java**). תוכן הקובץ יופיע בחלון הימני.
- בחלון הימני תוכלו לראות את השורה הראשונה של קוד התוכנית (ללא הרשאת הגישה **public** שאותה עליכם להוסיף).
- עליכם לכתוב תוכנית שתציג את שמכם המלא. כדי לכתוב את התוכנית התבוננו שוב בדוגמה הבאה (המופיעה בפרק):

```
/**
 * This program prints the student's full name.
 */
public class StudentDetails{
    public static void main (String[] args){
        //Prints the student's first name
        System.out.println (" The student's first name is: Albert ");
        //Prints the student's last name
        System.out.println (" The student's last name is: Einstein ");
    } //end of main
} //end of class
```

אם שמך המלא הוא אלפרד היציקוק, התוכנית תדפיס את הפלט הבא:

The student's first name is: Alfred

The student's last name is: Hitchcock

אל תשכחו לתעד את הקוד בצורה מספקת (בדומה לתיעוד המופיע בדוגמה). לאחר שסיימתם לכתוב את קוד התוכנית, שמרו את הקובץ שכתבתם על ידי בחירה בפעולה **Save** מתוך התפריט **File**.

## הידור תוכנית

כדי להדר את התוכנית לחצו על **F7** או בחרו באפשרות **Build Project** מהתפריט **Build**.



אם הפקודה מצליחה, המהדר יוצר קובץ שאותו ניתן להריץ (עם הסימט class, קובץ זה אינו נראה על המסך).

בחלון התחתון של סביבת הפיתוח תופיע הודעה כי הפעולה הסתיימה (Process completed).

אם ההידור נכשל, תופיע בחלון התחתון רשימה של שגיאות שנמצאו על ידי המהדר, ובסופה תופיע ההודעה Process completed. השורה הראשונה בתוכנית שגרמה לשגיאה תהיה ראשונה ברשימת השגיאות.

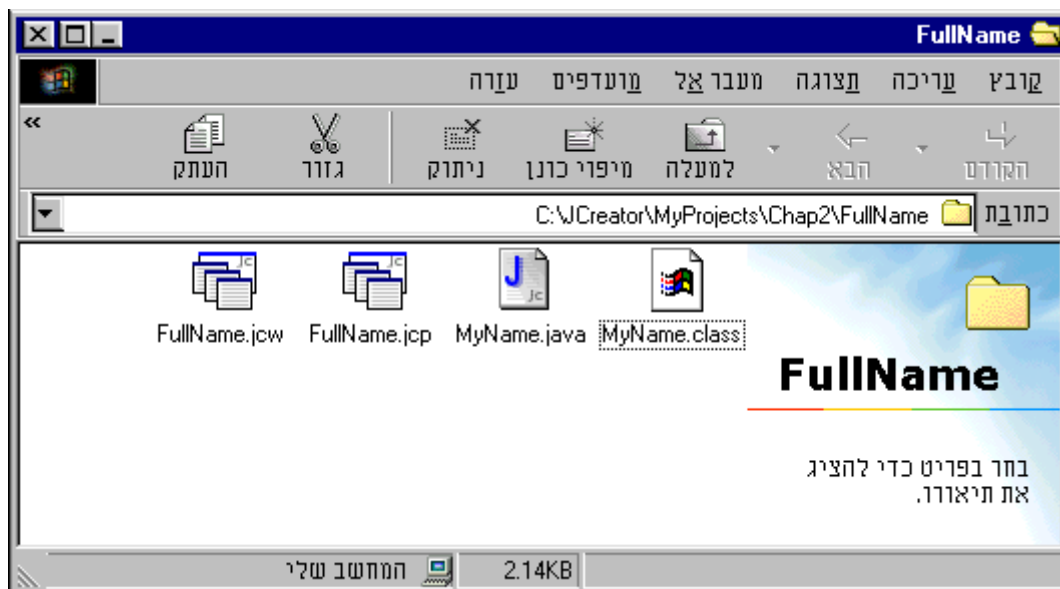
שימו לב, בחלון התחתון תופיע שורת הקוד השגויה ומתחת למילה שגרמה לשגיאה יופיע חץ קטן. אם יש שגיאות בתוכניתכם, תקנו אותן, שמרו את הקובץ המתוקן ונסו להדר פעם נוספת.

### הרצת תוכנית

כדי להריץ את התוכנית לחצו על F5 או בחרו באפשרות **Execute Project** מהתפריט **Build**. השוו את הפלט על המסך עם הפלט המבוקש. האם הפלט זהה לזה הנדרש? אם לא, תקנו את התוכנית בהתאם.

### הבנת התוכן של תיקיית הפרויקט

נפתח את תיקיית הפרויקט C:\JCreator\MyProjects\Chap2\FullName ונראה את תוכנה. התיקייה כוללת שני קבצים שנוצרו ברגע שנוצר פרויקט: FullName.jcp ו-FullName.jcw. מטרתם היא לשמור את כל האינפורמציה ששייכת לפרויקט. כמו כן נמצאים בתיקייה קובץ הגיאוה שכתבנו (MyName.java) וההידור שלו MyName.class.



## סגירת פרויקט

1. שמרו את קובצי הגיאווה על ידי בחירה בפקודה **Save All** מהתפריט **File**.
2. סגרו את אזור העבודה על ידי בחירה בפעולה **Close Workspace** מהתפריט **File**.

## פתיחת פרויקט קיים

1. בחרו בפעולה **Open Workspace** מהתפריט **File**.
2. נפתחת תיבת דו-שיח שכותרתה **Open**.
3. אתרו את תיקיית הפרויקט שלכם (שם התיקייה זהה לשם הפרויקט).
4. בתיקיית הפרויקט אתרו את קובץ "אזור העבודה" (**Workspace**) והקליקו עליו פעמיים. הסיומת של קובץ "אזור העבודה" היא **.jcw**.
5. הפעולה שביצעתם בשלב 4 פותחת את אזור העבודה שבו ממוקם הפרויקט שלכם. לאחר פתיחת אזור העבודה תראו על המסך את הפרויקט ואת כל הקבצים הכלולים בו.

**בהצלחה!**

## פרק 2 דף עבודה מס' 2

# פיצוח קוד של מנעול

### מטרות

1. עבודה ראשונית עם JCreator.
2. תרגול לולאות.

### מה עליכם לעשות?

דמיינו לעצמכם שיש בידכם מנעול, שעל מנת לפתוח אותו עליכם לדעת צירוף של שלושה מספרים. אתם מעוניינים לפתוח את המנעול, אך אינכם יודעים מהו צירוף המספרים הנחוץ, וגם אין בידכם כל כלי עבודה, שיוכל לעזור לכם לפתוח את המנעול בדרך אחרת. כתבו תוכנית המדפיסה את כל הצירופים האפשריים לפתיחת המנעול. לאחר שתהיה בידכם הדפסה שכזו, תוכלו לסמן כל צירוף שתנסו על הפלט, ותמנעו מבדיקת אותו צירוף מספרים יותר מפעם אחת. הניחו שהקוד לפתיחת המנעול הוא צירוף של שלושה מספרים ושכל מספר בצירוף יכול להיות בין 0 ל-2. למשל, הצירוף 2,1,0 הוא צירוף אפשרי.

**בהצלחה!**



## פרק 2 דף עבודה מס' 3

### שבע בום

#### מטרות

1. עבודה ראשונית עם JCreator.
2. תרגול לולאות.

#### מה עליכם לעשות?

עליכם להדפיס את כל המספרים מ-1 עד 100, כך שבמקום המספרים שמתחלקים ב-7 ללא שארית תודפס כוכבית. אחרי כל כוכבית ההדפסה תרד שורה. ההדפסה תראה כך:

1 2 3 4 5 6 \*

8 9 10 11 12 13 \*

...

אמנם במשחק המקורי גם מספרים שמכילים את הספרה 7 (כגון 17) הם "בוים", אבל לצורך הפשטות ויתרנו עליהם (מי שרוצה בכל זאת, מוזמן לנסות, זה אפשרי).

**בהצלחה!**

## פרק 2 דף עבודה מס' 4

### פעולות קלט ופלט

#### מטרות

1. שילוב ספרייה בפרויקט.
2. קבלת נתונים מהמשתמש (שימוש בחלון קלט).
3. הצגת פלט בחלון גרפי.
4. תרגול כללי התחביר של ג'אווה ומוסכמות הכתיבה בג'אווה.

#### רקע

אף שהדפסת פלט בג'אווה נעשית בצורה נוחה וקלה באמצעות הפקודה `System.out.print()`, שבה כבר השתמשנו, אין בג'אווה דרך המאפשרת קבלת קלט מהמשתמש בצורה פשוטה. על מנת לבצע קריאת נתונים מהמשתמש יהיה עליכם להשתמש בשיטות של מחלקה שנכתבה עבורכם: `InputRequestor`. בנוסף, נשתמש במחלקה `OutputWindow` כדי להציג פלט גרפי. מחלקות אלו אינן חלק משפת ג'אווה התקנית, אך הוספו לתרגילי המעבדה, כדי שתוכלו לבצע קריאת קלט בפשטות.

מחלקות אלו מובאות בתוך ספרייה בשם `io.zip` הנמצאת בתיקייה `C:\JCreator\MyProjects\InputOutput`. ספרייה היא אוסף מחלקות מהודרות (עם סיומת `.class`) אשר כווצו לקובץ מכווץ (`.zip` או `.jar`). על מנת להשתמש בספרייה יש צורך לשלבה בפרויקט. אופן צירוף ספרייה לפרויקט מתואר במדריך הטכני בסעיף 2.2.4.

לפני שתתחילו בעבודה קראו היטב את כל ההסברים המופיעים בדף עבודה זה.

#### קבלת קלט: המחלקה `InputRequestor`

על מנת לקבל משתנים מטיפוסים שונים עליכם להשתמש במחלקה `InputRequestor`. במחלקה זו אנו פונים לשיטות בעזרת סימון-הנקודה: שם המחלקה, נקודה, ואחריה שם השיטה. השיטות של `InputRequestor` מחזירות ערכים. נראה כמה דוגמאות:

- לקבלת קלט שהוא מספר שלם:

```
int i = InputRequestor.requestInt("...");
```

- לקבלת קלט שהוא מספר עשרוני:

```
double d = InputRequestor.requestDouble("...");
```

או:

```
float f = InputRequestor.requestFloat("...");
```

- לקבלת קלט שהוא מחרוזת:

```
String s = InputRequestor.requestString("...");
```

הסימון "..." בדוגמאות לעיל מציין שיכולה להופיע שם כל מחרוזת שהיא, בדרך כלל נשתמש במחרוזות בסגנון "Please enter a number". כל אחת מהפקודות הללו (`requestInt(...)`, `requestDouble(...)` וכו') גורמת ליצירת חלון, ובו כיתוב שתוכנו המחרוזת שהעברנו לפקודה כפרמטר. המחרוזת שבחרנו מודפסת על ידי השיטה בשורה אחת (אין אפשרות לעבור לשורה אחרת). הערך שיתקבל כקלט מהמשתמש יוכנס למשתנה המתאים (בדוגמה הראשונה – למשתנה `int i`, ובשנייה – למשתנה `double d` וכו').

## הצגת פלט: המחלקה `OutputWindow`

ראינו שכדי להציג פלט טקסטואלי ניתן להשתמש בשיטה `System.out.print()`. נציג כעת דרך המאפשרת הצגת פלט בחלון גרפי. על מנת להדפיס בחלון הגרפי יש לכתוב:

```
OutputWindow.print("...");
```

ואם רוצים לעבור שורה לאחר הדפסת הפלט:

```
OutputWindow.println("...");
```

`OutputWindow` יכול להדפיס מחרוזות ומשתנים מטיפוס בסיסי כלשהו. ניתן לנקות את המסך של `OutputWindow` על ידי הפקודה:

```
OutputWindow.clear();
```

## מה עליכם לעשות?

עליכם לכתוב מחלקה בשם `StudentGrades` שתכלול את השיטה `main(...)`. התוכנית תבקש מהמשתמש להקליד שלושה ציונים בזה אחר זה במקצועות: מתמטיקה, פיזיקה והיסטוריה. לאחר מכן, התוכנית תחשב את ממוצע הציונים ותציג את הפלט בחלון-פלט גרפי מטיפוס `OutputWindow`. הפלט יכיל את שלושת הציונים ואת הממוצע.

1. צרו פרויקט חדש בשם `StudentGrades`.
2. צרפו לפרויקט את הספרייה `io.zip`.
3. כתבו את המחלקה `StudentGrades` (בקובץ `StudentGrades.java`) וצרפו אותה לפרויקט.
4. הדרו את הפרויקט, והריצו אותו.

דרישות התוכנית:

1. הדפסות הקלט והפלט צריכות להיות קריאות ומובנות למשתמש.
2. תקינות הקלט: ערך הציון צריך להיות בין 0 ל-100. אם המשתמש מקליד ערך לא תקין התוכנית תבקש ממנו את הקלט שוב.

**בהצלחה!**

## פרק 2 דף עבודה מס' 5

### מחשבון פשוט

#### מטרות

1. תרגול מבני בקרה ולולאות, ותרגול נושאים תחביריים אחרים אשר נסקרו בפרק.
2. תרגול שילוב ספרייה בפרויקט.

#### מה עליכם לעשות?

כתבו תוכנית בשם SimpleCalculator, אשר תקבל מהמשתמש שני מספרים ולאחר מכן תבקש ממנו לבחור את הפעולה החשבונית שאותה הוא רוצה להפעיל על המספרים:

1. חיבור (addition)

2. חיסור (subtraction)

3. כפל (multiplication)

4. חילוק (division)

התוכנית תבצע על המספרים את הפעולה החשבונית שהמשתמש בחר, ותציג את התוצאה בחלון הפלט.

לדוגמה, אם המשתמש הקליד את המספרים 5 ו-11 ובחר בפעולה 2 (חיסור), התוכנית תדפיס:

The result is: -6

ואם המשתמש בחר בפעולה 4 (חילוק), התוכנית תדפיס:

The result is: 0.454545...

בסיום, התוכנית תשאל את המשתמש אם ברצונו להמשיך:

Press 1 to continue or any other digit to quit

אם המשתמש יקליד 1, התוכנית תחזור ותבקש ממנו שני מספרים ופעולה לביצוע. אם המשתמש יקליד מספר אחר, או אם הוא יסגור את החלון הגרפי, התוכנית תסתיים. המשתמש יכול לחזור על פעולות הקלדת המספרים ובחירת הפעולה החשבונית כמה פעמים שיחפוץ. לצורך כך, עליכם להשתמש בלולאה אינסופית.

עליכם לדאוג לתקינות הקלט!

לדוגמה, אם המשתמש בחר אפשרות של פעולה חשבונית שאינה קיימת, למשל, 0 או 6, התוכנית תבקש ממנו לשוב ולהקליד את בחירתו.

שימו לב: את ההסבר למשתמש לגבי הפעולות יש להציג בחלון פלט, ולא בחלון הקלט, כיוון שחלון הקלט אינו יכול להכיל הסברים בני יותר משורה אחת.

#### הערה חשובה

חלוקה של *int* ב-*int* מחזירה מספר שלם.

בקטע הקוד הבא למשל, התוצאה שתוצב במשתנה *c* תהיה 0.

```
...  
int a = 3;  
int b = 4;  
int c = a/b;
```

**בהצלחה!**



## פרק 2 דף עבודה מס' 6 מפרנהייט לצלזיוס

### מטרות

1. תרגול נוסף של צירוף ספרייה.
2. תרגול של מבני בקרה ולולאות.

### רקע

הנוסחה להמרת טמפרטורה מפרנהייט לצלזיוס היא:

$$\text{celsius} = 5 * (\text{fahrenheit} - 32) / 9$$

למשל, 104 מעלות בסולם פרנהייט שווה ל-40 מעלות בסולם צלזיוס. נקבל זאת כך:

$$40 = 5 * (104 - 32) / 9$$

### מה עליכם לעשות?

כתבו את התוכנית FahrenheitToCelsius, המציגה טבלת המרה של טמפרטורות הנתונות בסולם פרנהייט לטמפרטורות בסולם צלזיוס. תחילה התוכנית תקבל מהמשתמש שלושה מספרים שלמים: הטמפרטורה המינימלית להמרה (בפרנהייט), הטמפרטורה המקסימלית להמרה (בפרנהייט), גודל הקפיצה בין ערכי טמפרטורות הפרנהייט שיופיעו בטבלה. לדוגמה, אם המשתמש הקליד את הנתונים הבאים: טמפרטורה מינימלית = 10, טמפרטורה מקסימלית = 82, גודל קפיצה = 20, התוכנית תציג את הטבלה הבאה:

fahrenheit	celsius
10	-12
30	-1
50	10
70	21

יש לדאוג לתקינות הקלט: שלושת ערכי הקלט צריכים להיות מספרים שלמים, וערך הטמפרטורה המינימלית צריך להיות קטן מערך הטמפרטורה המקסימלית. הערכים שיוצגו אינם הערכים המדויקים בסולם צלזיוס, אלא החלק השלם שלהם. לדוגמה, אם על פי נוסחת ההמרה 0 מעלות פרנהייט הם 17.778- מעלות צלזיוס, הערך שיוצג הוא 17-.

**בהצלחה!**

## פרק 2 דף עבודה מס' 7

### אגדת המלך וגרגירי החיטה



#### מטרות

1. תרגול מבני בקרה ולולאות.
2. תרגול ההבדל בין טיפוסי הנתונים *long* ו-*int*.
3. תרגול שילוב ספרייה בפרויקט.

#### רקע

למדנו כי בשפת ג'אווה קיימים שישה טיפוסי נתונים בסיסיים עבור מספרים. כל טיפוס נתונים דורש נפח זיכרון שונה, ולכן מאפשר אחסון של מספרים בטווח ערכים שונה. שימוש בטיפוס נתונים שגוי עלול לגרור שגיאה תכנותית. לכן, עלינו להקפיד להשתמש בטיפוסים המתאימים לצרכינו. בתרגיל הבא תוכלו להיווכח שעלינו להקפיד על שימוש בטיפוסי נתונים נכונים אפילו בתוכניות פשוטות למדי.

#### אגדת המלך וגרגירי החיטה

לפני שנים רבות, בממלכה רחוקה, המציא איכר חכם אחד את משחק השח-מט. המשחק שעשה את המלך כל כך, עד שהכריז כי ייתן לממציאו פרס: "עד חצי המלכות ואת בתי אתן לו לאישה". הממציא, שהיה פיקח למדי וגם חמדן, לא הסתפק בהצעת המלך, ובמקומה ביקש את הפרס הבא:

"הוד מלכותך", אמר האיכר, "אני רק איכר צנוע המעבד את אדמתו, ולא אדע כיצד לנהל את מחצית הממלכה. לכן, אנא ממך, תן לי את הפרס הבא: על המשבצת הראשונה של לוח השח-מט הנח גרגיר חיטה אחד; על המשבצת השנייה של הלוח הנח שני גרגירים; ועל השלישית – פי שניים משהנחת על הקודמת. כך המשך עד שתגיע אל המשבצת האחרונה בלוח, ובגרגירים שיהיו על הלוח – אסתפק".

"איכר טיפש", חשב לעצמו המלך, וציווה להביא שק גרגירי חיטה, "על עושרה של מחצית הממלכה הוא ויתר בעבור כמה גרגירים". המלך התחיל להניח את הגרגירים על הלוח לפי בקשת האיכר, אולם מהר מאוד אזלו כל הגרגירים בשק הראשון, והמלך עוד לא כיסה אפילו את מחצית המשבצות שעל הלוח. הוא ציווה להביא שק שני ואחריו שלישי ועוד שק ועוד שק ועוד... במשבצת

ה-60, משאזלו שקי החיטה בממלכה כולה, הודה המלך בכישלונו, וכיוון שלא יכול היה לשלם לאיכר כפי שהבטיח לו, נתן לו את הממלכה כולה.

## מה עליכם לעשות?

כזכור, במשבצת ה-60 אזלו שקי החיטה בממלכה כולה. כמה גרגירי חיטה היו בממלכה? כתבו תוכנית בשם WheatCounter, המחשבת מספר זה ומדפיסה אותו. ההדפסה תיעשה בחלון הגרפי בעזרת המחלקה OutputWindow.

בכל פעם שהמלך מסיים להניח גרגירים על משבצת נוספת, עליכם להדפיס את מספר הגרגירים על הלוח כולו. דוגמת הדפסה:

1  
3  
7  
...

אף על פי שלא נשתמש בגרגירים אמיתיים, גם אנחנו נקלע למצוקה אם לא נשכיל להשתמש בטיפוס הנתונים הנכון עבור מספר גרגירי החיטה. כדי להיווכח בבעייתיות כתבו את התוכנית פעם עם משתנים מטיפוס *int* ופעם עם משתנים מטיפוס *long*.

## שאלה

מה קרה כאשר השתמשתם בכל אחד מהטיפוסים? מדוע?

בה3חה!

## פרק 2 דף עבודה מס' 8

### טעות לעולם חוזרת

#### מטרות

1. תרגול עבודה עם מהדר.
2. תרגול יצירת פרויקט בתיקייה קיימת.
3. תרגול צירוף קובץ קיים לפרויקט.

#### מה עליכם לעשות?

בתיקייה C:\JCreator\MyProjects\Chap2\Buggy נמצא קובץ בשם Buggy.java ובו שגיאות רבות. עליכם לתקן את הקובץ. לשם כך נסו להדר את הקובץ, וקראו בעיון את כל הודעות השגיאה שמתקבלות מהמהדר. הסבר על השפה של הקומפיילר ורשימת שגיאות נפוצות תוכלו למצוא במדריך הטכני סעיף 3 "הידור (קומפילציה)". בצעו את השלבים הבאים:

1. צרו פרויקט בשם Buggy. מיקום הפרויקט יהיה בתיקייה שבה נמצא הקובץ הנתון  
C:\JCreator\MyProjects\Chap2\Buggy : Buggy.java
2. צרפו לפרויקט את הקובץ Buggy.java. אופן צירוף קובץ לפרויקט מתואר במדריך הטכני בסעיף 2.2.2.
3. הדרו את הפרויקט, תקנו את כל השגיאות בעזרת הקומפיילר, והריצו את התוכנית.
4. איזו תוצאה התקבלה מהרצת התוכנית?

**בהצלחה!**

## פרק 2 דף עבודה מס' 9

### כלב יוצא לטיול

## מטרות

1. תרגול הלולאות: *while for*.
2. תרגול יצירת פרויקט בתיקייה קיימת.
3. תרגול צירוף קובץ קיים לפרויקט.
4. תרגול שילוב ספרייה בפרויקט.

## מה עליכם לעשות?

בפרויקט הזה יהיה עליכם להשתמש בקובץ שנכתב מראש עבורכם. הקובץ נמצא בתיקייה C:\JCreator\MyProjects\Doggie. קובץ, שלא כמו ספרייה, אינו מכוון, וכך אפשר לראותו בחלון ה-File View וניתן לעשות בו שינויים.

בדף עבודה זה עליכם להשלים שיטה בשם `go()`, שהיא חלק ממחלקה נתונה. על מנת לראות את כותרת השיטה שעליכם להשלים, בצעו את הפעולות הבאות:

1. צרו פרויקט חדש בשם Doggie (כלבלב) בתיקייה הקיימת: C:\JCreator\MyProjects\Doggie
2. צרפו לפרויקט את הספרייה Labyrinth.jar. אופן צירוף ספרייה לפרויקט מתואר במדריך הטכני בסעיף 2.2.4.
3. צרפו לפרויקט את הקובץ doggieLabyrinth.java (כלבלב במבוך). אופן צירוף קובץ לפרויקט מתואר במדריך הטכני בסעיף 2.2.2.

כעת, תוכלו לראות את קוד המחלקה DoggieLabyrinth.java, ובתוכה את כותרת השיטה `go()` שאותה עליכם להשלים (ראו איור 1).

```
//class DoggieLabyrinth inherit from class Labyrinth
public class DoggieLabyrinth extends Labyrinth {

    /**
     * Use the methods moveRight(), moveLeft(), moveDown(), moveUp()
     * to move doggie home and to eat all available food.
     */
    public void go() {
        //here you should write your code
    }

    public static void main(String[] args) {
        new DoggieLabyrinth().show();
    }
}
```

כאן עליכם להשלים את השיטה `go()`.

השיטה `show()` גורמת לפתיחת החלון הגרפי שאותו תראו בזמן הרצת התוכנית. שיטה זו משתמשת בשיטה `go()` שכתבתם.

איור 1

בשורת הקוד הראשונה במחלקה מופיעה המילה השמורה *extends*. מילה זו מורה על כך שהמחלקה DoggieLabyrinth יורשת מהמחלקה Labyrinth. על המשמעות המלאה של מנגנון הירושה ועל אופן השימוש במילה *extends* נדון בפרק "ירושה ופולימורפיזם".

שורת הקוד האחרונה במחלקה קוראת לשיטה `show()`, הגורמת לפתיחת החלון הגרפי שאותו תראו בזמן הרצת התוכנית. שיטה זו משתמשת בשיטה `go()` שאתם תכתבו.

### המשימה: לאכול את כל הנקניקיות בדרך הביתה

בפינה השמאלית-העליונה של הלוח המופיע באיור 2 תוכלו לראות את הכלבה "נומה". נומה מעוניינת לחזור הביתה (אל המשבצת שעליה כתוב Home), תוך אכילת כל הנקניקיות שהיא מוצאת בדרכה. כדי לעזור לנומה להשיג את מטרתה עליכם להשלים את השיטה `go()`, הגורמת לנומה ללכת בין בתי השכונה תוך איסוף כל הנקניקיות שבדרכה. ממשו את השיטה `go()` באמצעות שש השיטות הבאות:

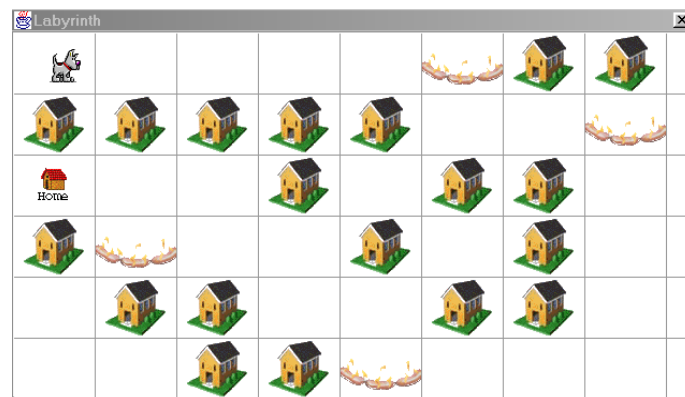
שם השיטה	תיאור השיטה
<code>void moveRight()</code>	ללכת צעד אחד ימינה (משבצת אחת בלוח)
<code>void moveLeft()</code>	ללכת צעד אחד שמאלה (משבצת אחת בלוח)
<code>void moveDown()</code>	ללכת צעד אחד למטה (משבצת אחת בלוח)
<code>void moveUp()</code>	ללכת צעד אחד למעלה (משבצת אחת בלוח)
<code>void eat()</code>	לאכול נקניקייה אחת
<code>boolean isFood()</code>	השיטה בודקת אם נשאר אוכל במשבצת שעליה עומדת הכלבה אם אין אוכל השיטה תחזיר <code>false</code> אם יש אוכל השיטה תחזיר <code>true</code>

הטבלה לעיל היא דוגמה של הגדרת **ממשק** שמאפשרת שימוש בשיטות שמישהו אחר מימש.

שימו לב, בלוח שלפניכם יש שלושה סוגים של משבצות: משבצות עם בתים, משבצות ריקות ומשבצות עם מזון (נקניקיות). מיקום הבתים והמזון בלוח הוא קבוע.

- נומה יכולה לעבור דרך שני סוגים של משבצות בלבד: משבצות ריקות ומשבצות עם מזון. היא אינה יכולה לעבור דרך משבצות בהן יש בתים.
- כמות המזון בכל משבצת מזון אינה ידועה.

כדי להוליך את נומה במסלול הנכון הביתה, התבוננו באיור 2, ותכננו את המסלול כך שנומה תאכל את כל הנקניקיות שבדרכה.



השלימו את השיטה `go()` כנדרש. לאחר שהשלמתם את השיטה שמרו את הפרויקט, הדרו אותו והריצו אותו.

**בהצלחה!**

## פרק 2 דף עבודה מס' 10

### מסע בין כוכבים

#### מטרות

1. תרגול מבני הבקרה *while* ו-*switch*.
2. תרגול יצירת פרויקט בתיקייה קיימת.
3. תרגול צירוף קובץ קיים וספריות בפרויקט.

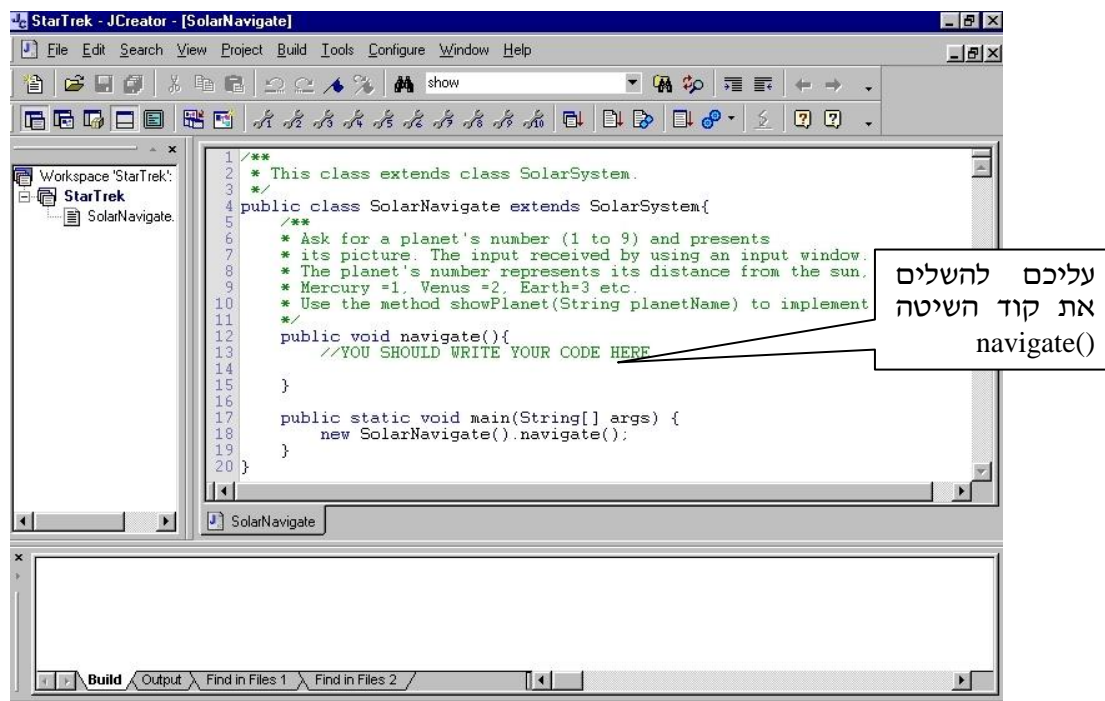
#### מה עליכם לעשות?

הקבצים הנדרשים לביצוע דף עבודה זה נמצאים בתיקייה `C:\JCreator\MyProjects\StarTrek`. בדף עבודה זה עליכם להשלים שיטה בשם `navigate()`, שהיא חלק ממחלקה נתונה בשם `SolarNavigate`. על מנת לראות את כותרת השיטה שעליכם להשלים, בצעו את הפעולות הבאות:

1. צרו פרויקט חדש בשם `StarTrek` (מסע בין כוכבים). מקמו פרויקט זה בתיקייה הקיימת:  
`C:\JCreator\MyProjects\StarTrek`
2. צרפו לפרויקט את קובץ הספרייה `Stars.zip`. אופן צירוף ספרייה לפרויקט מתואר במדריך הטכני בסעיף 2.2.4.
3. צרפו לפרויקט את קובץ המחלקה `SolarNavigate.java`. אופן צירוף קובץ לפרויקט מתואר במדריך הטכני בסעיף 2.2.2.



כעת, תוכלו לראות את קוד המחלקה SolarNavigate.java, ובתוכה את כותרת השיטה navigate() שאותה עליכם להשלים (ראו איור 1).



בשורת הקוד הראשונה במחלקה מופיעה המילה השמורה *extends*. מילה זו מורה על כך שהמחלקה SolarNavigate יורשת מהמחלקה SolarSystem. על המשמעות המלאה של מנגנון הירושה ועל אופן השימוש במילה *extends* נדון בפרק "ירושה ופולימורפיזם". שורת הקוד האחרונה במחלקה פותחת את החלון הגרפי שאותו תראו בזמן הרצת התוכנית, ומשתמשת בשיטה navigate() שאתם תכתבו.

### המשימה: ניווט בין כל כוכבי הלכת

נמספר את כל כוכבי הלכת על פי מרחקם מהשמש. כוכב הלכת הקרוב ביותר אל השמש יקבל את המספר 1, והרחוק ביותר את המספר 9. לדוגמה, מספרו של כדור הארץ הוא 3 ומספרו של כוכב הלכת הרחוק ביותר מהשמש, פלוטו, הוא 9. רשימה של כל כוכבי הלכת לפי מרחקיהם מהשמש תוכלו למצוא בסוף דף העבודה. עליכם לממש את השיטה navigate(). השיטה תקבל מהמשתמש מספר המייצג את אחד מתשעת כוכבי הלכת ותציג את התמונה שלו.

### הערה חשובה

השיטה navigate() צריכה לאפשר למשתמש לחזור ולהקליד מספר המייצג כוכב לכת כמה פעמים שהוא רוצה. לכן, עליכם לחזור ולבקש מהמשתמש קלט, גם לאחר שהצגתם את התמונה של כוכב הלכת שהמשתמש בחר. התוכנית תסתיים כאשר המשתמש יסגור את חלון הקלט או את החלון שבו מוצגת תמונת הכוכב. לצורך כך, עליכם להשתמש בלולאה אינסופית. ממשו את השיטה navigate() באמצעות השיטה הבאה:

שם השיטה	תיאור השיטה
<code>void showPlanet(String planetName)</code>	מציגה תמונה של כוכב הלכת המתאימה לשם הכוכב שהועבר כפרמטר (ראו פירוט בהמשך)

דוגמאות לשימוש בשיטה `showPlanet`:

קריאה לשיטה עם הפרמטר `Earth`:

```
showPlanet("Earth");
```

תציג תמונה של כוכב הלכת "ארץ".

קריאה לשיטה עם הפרמטר `Mars`:

```
showPlanet("Mars");
```

תציג תמונה של כוכב הלכת "מאדים".

רשימת כוכבי הלכת לפי מרחקם מהשמש (מהקרוב ביותר אל הרחוק ביותר):

Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto

## שאלה

האם להשלמת השיטה `navigate()` השתמשתם בפקודה `if-else` או בפקודה `switch`? הסבירו את בחירתכם.

**בהצלחה!**