מבוא למבוא למדעי המחשב

לקראת הקורס "מבוא למדעי המחשב"
המרכז הבינתחומי הרצליה

10/3/2010

בועז קנטור

תוכן

מהו מחשב? 4

חומרה 4

ה CPU 4

הזכרון 4

לוח האם 5

תוכנה 5

השפה הבינארית 5

מערכת ההפעלה ושפות עיליות 5

הידור 6

כל התהליך 6

איך מתכנתים? 7

התכנית הראשונה 7

סביבת העבודה 7

סביבת פיתוח 8

מקמפלים ג'אווה 9

מריצים ג'אוה 9

שינויים בתכנית 9

שימוש בספריות הקורס 10

טעימה מהחומר 10

שימוש בזכרון 10

הקצאת מקום בזכרון 11

עדכון ערכים במקום שהוקצה 11

קריאת ערכים מהזכרון 11

התנייה בתכנית 12

לולאה בתכנית 13

קיבוץ פקודות 14

תרגיל מסכם 14

עזרה 15

מהו מחשב?

אנו מכירים סוגים רבים של מחשבים: מחשב שולחני, מחשב נייד, מחשבים גדולים של בנקים ואפילו מחשבים קטנים בתוך הטלפונים הסלולריים שלנו. אבל ממה עשויים כל המחשבים האלה?

מחשב טיפוסי מורכב משלושה דברים:

1. חומרה (Hardware): זהו החלק הפיסי של המחשב. המתכת, הפלסטיקים והמעגלים המודפסים שמוחבאים בתוכו.
2. תוכנה (Software): תוכנה אינה מוחשית, ואי אפשר לגעת בה. תוכנה היא למעשה אוסף של הוראות, כמו מתכון בישול, הכתובות בשפה שהחומרה יכולה להבין.
3. משתמשים (Users): אמנם לא ניתן לקנות משתמשים בחנות המחשבים, אך אין למחשב זכות קיום בלי האנשים שישתמשו בו.

ישנם אנשים שמשתמשים בתוכנה מצד אחד, ואנשים אחרים שכותבים את התוכנה (מתכנתים, אנחנו!) מצד שני.

חומרה

מה שמתחבא בתוך קופסת הקרטון שיוצאת ממפעל המחשבים נקרא "חומרה". נבדיל בין שני סוגים של חומרת מחשב: המחשב עצמו והציוד ההיקפי (Peripherals). המחשב עצמו הוא אותו מארז מברזל שיושב על הריצפה או השולחן בסלון. הציוד ההיקפי הוא מה שמתחבר לקופסה הזאת: עכבר, מקלדת, מסך, Disk-on-key, רמקולים וכיו"ב. לא נתמקד בציוד ההיקפי, אלא במחשב עצמו. החלקים בהם נתמקד כעת הם ה CPU, הזכרון ולוח האם.

ה CPU

בכדי שהמחשב יוכל להריץ תוכנות, יושב בליבו צ'יפ, שבב קטן, שנקרא "יחידת עיבוד מרכזית" (CPU). אמנם השבב הזה קטן מאוד, אך בעבר יחידות העיבוד היו בגודל מקרר. יחידת העיבוד מורכבת בעצמה מחלקים קטנים יותר, עליהם תלמדו בקורסים מתקדמים יותר. היחידה יודעת לבצע הוראות פשוטות, כמו חישובים מתמטיים או לזכור מספרים. דרך העבודה עם ה CPU היא עקרון מאוד בסיסי במחשבים, שחשוב לזכור אותו: קלט ← עיבוד ← פלט (input→processing→output). הקלט הוא ההוראה שנשלחת ליחידת העיבוד, יחד עם הפרמטרים הנדרשים (כמו המספרים שצריך לחשב). היחידה מעבדת את ההוראה, למשל מחשבת תוצאת חיבור של שני מספרים, והפלט הוא התוצאה של החישוב. כך, למשל, בכדי לחשב 4*10 (הדרך המקובלת לתאר את פעולת הכפל במחשבים היא על ידי הכוכבית * (Asterisk)), נשלח לקלט של יחידת העיבוד את ההוראה "כפול 4 ב 10". היחידה תבצע את החישוב, ותשלח לפלט את התוצאה 40.

הזכרון

מה קורה כשנדרשת פעולה יותר מסובכת מארבע כפול עשר? למשל, 3+10*4? יחידת העיבוד לא יודעת לבצע חישוב של יותר משני מספרים, בדיוק כמונו. איך אנחנו פותרים חישוב כזה? אנחנו נכפול 4 ב 10, נזכור בראש 40, ונוסיף למה שזכרנו את המספר 3. כך בדיוק עובד ה CPU: הוא מחשב את הכפל ו"זוכר" את התוצאה, ורק אז מוסיף לה 3. מה זאת אומרת "זוכר"? לשם כך הומצא הזכרון. הזכרון מאפשר למחשב לבצע פעולות מורכבות. אך לזכרון יש שימושים נוספים: סוגים שונים של זכרון משמשים לצרכים שונים. ראשית, נגדיר שלושה מאפיינים של זכרון: מהירות גישה, נפח ונדיפות (volatility). מהירות גישה היא התשובה לשאלה "כמה מהר ניתן לכתוב לזכרון, וכמה מהר ניתן לקרוא ממנו". נפח הוא התשובה לשאלה "כמה נתונים ניתן לשמור בזכרון", ונדיפות היא התשובה לשאלה "האם הזכרון ימשיך לזכור את הנתונים שבו גם אחרי שנכבה את המחשב". לצורך הלימוד הבסיסי נתמקד בשלושה סוגי זכרון: זכרון פנימי של ה CPU, זכרון ראשי של המחשב ומדיה חיצונית:

- זכרון פנימי של ה CPU הוא קטן מאוד, ונמצא בתוך ה CPU. זהו הזכרון המהיר ביותר מבין שלושת הסוגים, אך המהירות באה על חשבון שני המאפיינים האחרים: הנפח שלו קטן מאוד והוא נדיף, כלומר, תוכנו נמחק עם כיבוי המחשב. על הזכרון הזה, הנקרא "רגיסטרים" (registers) תלמדו בקורסים מתקדמים יותר.
- זכרון ראשי של המחשב, ה RAM, איטי יותר מהזכרון הפנימי, וגם הוא נדיף, אך הנפח שלו יכול להיות גדול מאוד. הזכרון הראשי מוכר לכם כמאפיין של מחשב בעת קנייה "המחשב מגיע עם 2 גיגה בייט" משמעו: ניפחו של הזכרון הוא 2 מיליארד בייט. האם זה באמת 2 מיליארד? מהו בייט בכלל? על השאלות האלה נענה בהמשך.

- מדיה חיצונית היא זיכרון שמאפשר אגירה של נפחים גדולים מאוד, בלי הגבלה, למעשה, אך הוא זכרון איטי מאוד. היתרון הגדול של המדיה החיצונית הוא בכך שהיא אינה נדיפה, כלומר, בכדי שהמידע לא יימחק מהזכרון אין צורך במקור חשמל. רוב סוגי המדיה החיצונית מוכרים לכם. היא נשמרת בצורה של דיסקים מגנטיים (דיסק קשיח, למשל), דיסקים אופטיים (CD, DVD) או זכרון Flash (Disk on key).

לוח האם

מלבד יחידת העיבוד המרכזית והזכרון, המחשב מורכב מהמון רכיבים נוספים. בכדי שכל הרכיבים יוכלו "לדבר" האחד עם השני, הם כולם מורכבים על בסיס אחד: לוח האם (motherboard). לוח האם הוא משטח עליו מורכבים פסי מתכת המהווים ערוצי תקשורת בין הרכיבים, כשבקצוות פסי המתכת מורכבים הרכיבים עצמם: המעבד, הזכרונות השונים, החיבור לציוד ההיקפי, ועוד. אם תרצו, לוח האם הוא כמו השילדה של מכונית.

את כל החומרה שלמדנו כעת אורזים בתוך קופסת מתכת שנקראת "מארז". בשנים האחרונות הצליחו למזער את כל החלקים ולסדר אותם בצורה אופטימלית בתוך קופסה קטנה יותר וניידת יותר, וכך נולד המחשב הנייד (Laptop/notebook computer).

תוכנה

כאמור, התוכנה היא אוסף של הוראות עם פרמטרים. האם Facebook הוא תוכנה? האם "שלח מייל" היא הוראה של תוכנה? בגדול ניתן לומר שכן, אך על מנת להבין טוב יותר מהי תוכנה, נתחיל דווקא ביחידת העיבוד המרכזית. בואו ננסה לתכנת הוראה של "חשב 3+10*4".

השפה הבינארית

ה CPU לא יודע באמת לקרוא מתכון בישול. הוא גם לא מבין הוראות בעברית, ואפילו לא באנגלית. למעשה, בגלל שהוא מורכב ממעגלים חשמליים, הוא מבין רק שתי אותיות: אפס ואחת. לשפה המורכבת משתי האותיות האלה קוראים "השפה הבינארית" (Binary) והיא "שפת המכונה" של המחשב. כאשר נזרים למעבד מתח חשמלי, הוא יבין שזה האות 1, וכאשר המתח יורד, הוא מבין שזה 0. אז איך נבקש ממנו לכפול 10*4? נרכיב את המספר 4 מצירופים שונים של 0 ו 1, וכך גם את המספר 10. אמנם המספר 10 מורכב מאפס ואחת, אך הייצוג שלו בשפה הבינארית אינו 10, אלא 1010. לפקודה "כפל" גם יש ייצוג בשפה הבינארית, המורכב מצירוף של אפסים ואחדות. על השפה הבינארית וכיצד ממירים מספרים מהשיטה העשרונית שאנו רגילים אליה לשיטה הבינארית, תלמדו לעומק בתחילת הקורס הזה, כמו גם המרות בין שיטות ספירה נוספות.

מערכת ההפעלה ושפות עיליות

אז האם בשביל לדבר עם המחשב אני צריך לדעת "לדבר" בינארית? באופן מפתיע, פעם באמת דיברו כך עם המחשב. במקום מקלדת היו כרטיסי קרטון עם חורים שסימלו 0 ו 1. אך היום זה קל הרבה יותר. בשביל להתגבר על הקושי שבהתמודדות עם השפה הבינארית המציאו את מערכת ההפעלה ושפות עיליות. מערכת ההפעלה היא תוכנה בפני עצמה, שיודעת לדבר עם ה CPU. מה זאת אומרת "לדבר עם ה CPU"? היא יודעת לקבל פקודה מהתוכנה שלנו, לתרגם את הפקודה לשפה הבינארית, ולשלוח את הפקודה ל CPU. היא גם יודעת לתרגם את הפלט (זוכרים?) של ה CPU

לשפה שהתוכנה שלנו מכירה. מערכות הפעלה ידועות הן "חלונות" של חברת מיקרוסופט ולינוקס. אילו עוד מערכות הפעלה אתם מכירים?

אז איך התוכנה שלנו תשלח "4*10+3" למערכת ההפעלה? איך אנחנו יכולים בכלל לדבר עם מערכת ההפעלה? באנגלית? מה עם כל השפות ששמענו עליהן, כמו ג'אווה, C# ו Visual Basic? מערכת ההפעלה לא באמת יודעת אנגלית, והיא גם לא מבינה שפות עיליות. היא כן מבינה שפת מכונה, אותה קשה ללמוד. שפת תיכנות עילית היא שפה שהמציאו בשביל להקל עלינו, המתכנתים, בכתיבת תוכנה. השפה מורכבת מהוראות בשפה האנגלית, שהן די מובנות ואינטואיטיביות. אחרי שכתבנו תוכנה בשפה האנגלית, כל שנותר לנו הוא להריץ תוכנות שיתרגמו את הפקודות שלנו לפקודות שמערכת ההפעלה יודעת להעביר למחשב.

הידור

בקורס הזה נלמד את אחת השפות הנקראת ג'אווה (Java). ג'אווה היא שפה עילית, ואוסף המילים ממנה היא מורכבת, כמו גם התחביר שלה, דומים לשפה האנגלית, מה שיקל עלינו בכתיבת תוכנות. כך, למשל, בשביל להדפיס למסך את התוצאה של הביטוי 4*10+3, נכתוב תוכנה שנראית בערך כך:

```
System.out.println(4*10+3);
```

אינטואיטיבי, נכון? איך מתרגמים את האנגלית הזאת עבור מערכת ההפעלה? לשם כך קיים המהדר (Compiler). המהדר הוא תוכנה, היודעת לקרוא את הפקודות שלנו שנכתבו בשפה עילית, ולתרגם אותן לשפת מכונה. את התוצאה של הקומפילציה שומר המהדר בתוך קובץ, אותו יודעת מערכת ההפעלה לקרוא (קבצים עם סיומת exe. הם כאלה), לתרגם לשפה הבינארית, ולשלוח למעבד.

כל התהליך

זהו תרשים המתאר את התהליך הקלאסי של תכנות, הידור והרצה. בג'אווה (וכן בשפות Net. השונות) קיים תהליך ביניים, עליו נלמד במהלך הקורס.

איך מתכנתים?

התכנית הראשונה

בואו נסתכל על התכנית הבסיסית הבאה (מספרי השורות אינם חלק מהתכנית):

```
[1]   public class MyProgram {
[2]         public static void main(String[] args) {
[3]               System.out.println(4*10+3);
[4]         }
[5]   }
```

זוהי תוכנית שלמה הכתובה בג'אווה. כאשר נריץ אותה (אחרי שעשינו מה?), היא תדפיס למסך את המספר 43.

במהלך הקורס תלמדו על כל חלקי התכנית, אך בשלבים הראשונים נתמקד בשלושה חלקים בלבד:

- שם התכנית: בשורה [1] כתוב השם MyProgram. זהו שם התכנית שלנו. לג'אווה לא משנה מהו שם התכנית, כל עוד שם הקובץ זהה לו לחלוטין. למשל, תכנית זו צריכה להיות כתובה בתוך קובץ שנקרא MyProgam.java. אתם יכולים לבחור איזה שם שאתם רוצים, כל עוד הוא עונה על חוקי השמות של ג'אווה (כמו למשל, אסור שיהיה בו רווח), וששם הקובץ זהה לשם שבחרתם (עם הסיומת java.).
- האלגוריתם: שורה [3] היא הוראה בג'אווה להדפיס למסך. זהו בעצם הפתרון לבעיה שהצגנו בפרק הקודם. הדרך שבה אנו בוחרים לפתור את הבעיה שלנו נקרא "אלגוריתם". אלגוריתם הוא סדרה של הוראות, שאם יתבצעו לפי הסדר הן יפתרו לנו בעיה כלשהי.
- "סימני פיסוק": כמו בכל שפה, גם בג'אווה יש חוקי דקדוק הכוללים מעין "סימני פיסוק". נבחן שלושה מרכזיים:
  - נקודה-פסיק ; - נקודה-פסיק (באנגלית: semicolon) מסמלת סוף של פקודה בשפה. כל פקודה חייבת להסתיים בנקודה-פסיק, למעט מקרים שדורשים סוגריים-מסולסלים (ראה בהמשך).
  - סוגריים מסולסלים { } – סוגריים מסולסלים מחברים מספר פקודות של ג'אווה לקבוצה אחת. מתי זה טוב? למשל, כאשר יש לנו מספר פקודות שנרצה להתייחס לכולן ביחד כתת-פתרון. פקודה שמסתיימת בסוגריים מסולסלים לא צריכה נקודה-פסיק בסופה.
  - סוגריים () – הסוגריים הבאים אחרי פקודה (בשורה [3] אחרי הפקודה `println`, למשל), הם האיזור שבו אנו מגדירים פרמטרים משלנו לפקודה. ההבדלה בין פקודה לפרמטר היא חשובה ביותר; הפקודה היא ההוראה עצמה, ואילו הפרמטר הוא נתונים נוספים שאתם בוחרים, שההוראה צריכה לקחת בחשבון בזמן שהיא מתבצעת. כך, למשל, "פנה ימינה" היא פקודה לפנות, עם הפרמטר "ימינה", לעומת "פנה שמאלה" שהיא אותה הפקודה אך עם פרמטר אחר. קריאה לפקודה ללא פרמטרים בג'אווה עדיין דורשת כתיבת סוגריים, אלא שנשאיר אותם ריקים.

כעת, משהבנו איך התוכנית בנויה ומה היא עושה, נוכל לבדוק אם היא עובדת.

סביבת העבודה

ראשית עלינו ליצור סביבת עבודה. לשם כך בצעו את ההוראות שבמסמך שבאתר. לאחר שהתקנו והגדרנו את ג'אווה על המחשב שלנו, נעקוב אחרי האלגוריתם הבא ליצירת מחיצה בדיסק הקשיח שבה נעבוד:

1. ליחצו על הכפתור Start. מיד ייפתח לנו תפריט אפשרויות.
2. (רק אם יש לכם Windows XP): ביחרו באפשרות "Run".
3. כעת נכתוב `cmd` ונלחץ ENTER.
4. אם הצלחתם עד כה, אמור להיפתח לכם חלון הפקודות של מערכת ההפעלה, שנראה כטקסט לבן על רקע שחור.
5. ניצור מחיצה בשם testing ע"י הרצת הפקודה mkdir:

```
mkdir \testing
```

6. כעת נעבור למחיצה שיצרנו על ידי הרצת הפקודה cd:

```
cd \testing
```

בהמשך הפרק נעבוד בתוך המחיצה הזאת, הנקראת `testing`.

בכדי לבדוק אם הצלחתם להתקין את ג'אווה, ורק בשביל לבדוק, כיתבו כעת את הפקודה הבאה:

```
java -version
```

אם קיבלתם את התוצאה הבאה:

```
'java' is not recognized as an internal or external command,
operable program or batch file.
```

אזי ג'אווה אינה מותקנת היטב על מחשבכם ועליכם לחזור על ההוראות שבמסמך. נסו תחילה לבצע מחדש רק את החלק שמגדיר את משתנה הסביבה בשם Path. אם הגדרתם מחדש את משתנה הסביבה, סיגרו את חלון ה cmd וחיזרו לשלב (1) כדי לפתוח אותו מחדש.

לעומת זאת, אם אחרי הרצת הפקודה קיבלתם תוצאה שנראית כך:

```
java version "1.6.0_10-rc"
Java(TM) SE Runtime Environment (build 1.6.0_10-rc-b28)
Java HotSpot(TM) Client VM (build 11.0-b15, mixed mode, sharing)
```

הרי שג'אווה מותקנת אצלכם ואתם מוכנים להתחיל לעבוד.

עדכון מספטמבר 2010: הגירסה הרשומה צריכה להיות 1.6.0_21, ולא כפי שכתוב.

סביבת פיתוח

אמנם התוכנית שכתבנו בתחילת הפרק קצרה, אך כתיבת תוכניות משמעותיות בג'אווה דורשת המון המון טקסט. במקרים כאלה מקובל לעבוד עם כלי עבודה שלתוכם מקלידים את התוכנית, והם עוזרים לנו לנהל את פיתוח התוכנה ממש כפרוייקט. בהמשך הקורס נעבוד עם אחד הכלים האלה שנקרא "eclipse". לבינתיים נסתפק בכתיבת הקובץ בצורה הפשוטה ביותר. לשם כך נכתוב:

```
notepad MyProgram.java
```

שימו לב ששם הקובץ חייב להיות זהה לחלוטין לשם התוכנית שכתובה בתוכו! משנפתח חלון העריכה, נישאל האם ברצוננו ליצור קובץ חדש בשם MyProgram.java. נענה לזה בחיוב, והתכנית notepad תיצור עבורנו את הקובץ הנ"ל

בתוך המחיצה testing. כעת נישאר עם חלון העריכה בלבד. נקליד לתוכו את התוכנית מתחילת הפרק ונשמור את הקובץ (ע"י לחיצה על CTRL-S או ע"י בחירה בתפריט File/Save). מרגע זה, עלינו לזכור לשמור כל שינוי שנעשה בקובץ.

מקמפלים ג'אווה

בשביל לקמפל תוכנית בג'אווה עלינו להשתמש בקומפיילר. נקמפל את התוכנית שלנו ע"י קריאה לקומפיילר עם שם הקובץ כפרמטר:

```
javac MyProgram.java
```

כעת יקרה אחד משלושת הבאים (בהנחה שהעתקתם את השורה בדיוק):

1. הקומפיילר לא ירוץ. לכך יכולות להיות 2 סיבות:
   a. סביבת העבודה שלכם לא מותקנת היטב. חיזרו על ההוראות שבמסמך שבאתר, וספציפית על החלק שמגדיר את משתנה הסביבה Path.
   b. לא קיים קובץ בשם MyProgram.java, כנראה בגלל שלא כתבתם את השם נכון כשיצרתם את הקובץ עם התכנית notepad, או שיצרתם אותו עם השם הנכון אך לא תחת המחיצה testing, או שאינכם נמצאים כעת במחיצה testing.
2. הקומפיילר ירוץ אך ייכשל ("התכנית לא תתקמפל"). נדע שהתכנית לא התקמפלה (או "לא עברה קומפילציה") אם קיבלנו כל מיני הודעות על שגיאות עם הסברים. ייתכן מאוד שלא העתקתם את התכנית בדיוק כפי שנכתבה. שימו לב גם לאותיות גדולות/קטנות באנגלית (casing). ג'אווה דורשת דיוק גם בחלק הזה, ולטעות ב casing זו שגיאה נפוצה.
3. הקומפיילר ירוץ, והתכנית תעבור קומפילציה בהצלחה. נדע שזה קרה אם לא קיבלנו שום הודעה.

אם הקומפיילר לא רץ (מקרה 1), עליכם לוודא שוב שסביבת העבודה שלכם תקינה. שימו לב שלאחר שעשיתם שינויים במשתני הסביבה (Path, CLASSPATH) עליכם לסגור את חלון ה cmd ולפתוח אותו מחדש.

אם התכנית לא עברה קומפילציה (מקרה 2), חיזרו שוב לקובץ, תקנו את התוכנית שלכם, שימרו את השינויים, ונסו לקמפל מחדש את התוכנית.

אם הצלחתם לקמפל את התכנית (מקרה 3), ורק במקרה זה, אתם יכולים לנסות ולהריץ אותה.

מריצים ג'אוה

נריץ את התכנית שלנו ע"י הרצת פקודה בשם java, כאשר הפרמטר יהיה שם התוכנית. נזכיר רק, ששם התוכנית זהה לשם הקובץ, רק בלי הסיומת .java:

```
java MyProgram
```

אם קיבלתם הודעת שגיאה, סביר להניח שלא הוגדר אצלכם משתנה CLASSPATH היטב (האם שכחתם לשים נקודה בערך המשתנה?). בעתיד נגלה שייתכנו סיבות רבות לכך שהרצת התוכנית גרמה להדפסה של הודעות שגיאה. באופן כללי, כאשר התוכנית רצה אך לא מבצעת מה שתכננו שהיא תעשה, אנו קוראים לשגיאה בשם "באג" (השם "באג", או בעברית "חרק" נקבע היסטורית כאשר הבאג הראשון נגרם באמת בעקבות חרק שנתקע במחשב).

אם הודפסה לכם תוצאת החישוב 4*10+3, אזי התוכנית רצה בהצלחה! כעת אתם מוכנים לכתוב כל תכנית ג'אווה, לקמפל ולהריץ אותה.

שינויים בתכנית

לפני סיום, בואו נוודא שאנו מצליחים גם לשנות את התוכנית בהצלחה. החליפו את החישוב 4*10+3 בטקסט הבא: "Hello world!" (כולל המרכאות). התוכנית תיראה כך:

```
[1]   public class MyProgram {
[2]         public static void main(String[] args) {
[3]               System.out.println("Hello, world!");
[4]         }
[5]   }
```

שימרו את השינויים, קמפלו, והריצו. כעת חישבו, מה גרם לכך שהתכנית האחרונה תכתוב את הטקסט שלנו בדיוק כפי שכתבתנו אותו, אך התוכנית הקודמת החליטה שעליה לבצע חישוב מתמטי במקום להדפיס את הטקסט המדויק? נסו גם להדפיס את הטקטס Hello, world! ושורה אחריו את תוצאת החישוב 43 באותה התוכנית. מה עליכם לעשות? רמז: מדובר בשתי פקודות println נפרדות. מה עליכם לעשות כדי להחליף את הסדר בין ההדפסות, כלומר שקודם יודפס 43 ובשורה אחר כך יודפס Hello, world! ?

שימוש בספריות הקורס

במהלך הקורס תקבלו מצוות הקורס ספריות תוכנה שונות. אלו הם קטעי קוד שנכתבו ע"י צוות הקורס, ונועדו לשימושכם; בין אם כדוגמאות במהלך ההרצאות ובין אם ספריות שתידרשו להשתמש בהן במהלך התרגילים. בכדי שתוכלו להשתמש בספריות האלה, תצטרכו להוריד אותן מאתר הקורס (עפ"י ההנחיות שיינתנו במהלך הקורס) ולקשר אותן לתכניות שלכן. חלק זה יכין אתכם לשימוש בספריות הקורס.

אמנם בחלקים הקודמים יצרנו מחיצה בכונן הקשיח על מנת לבדוק את סביבת העבודה, אך מומלץ שתהיה לכם מחיצה המיועדת לכל חומר הקורס (תרגילים, מצגות, בחינות וכדומה). אחת מתתי המחיצות של המחיצה הזאת תהיה מוקדשת לתרגילים שתפתרו. נניח שיצרתם ספריה בשם:

```
C:\IDC\Intro2CS\Exercises
```

תוכלו לפתור את התרגילים בתתי המחיצות:

```
C:\IDC\Intro2CS\Exercises\Ex01
C:\IDC\Intro2CS\Exercises\Ex02
.
.
```

בשלב הבא, צרו מחיצה מיוחדת המוקדשת לספריות הקוד שאנו נספק לכם. למשל:

```
C:\IDC\Intro2CS\Exercises\Libraries
```

בכדי שתוכלו להשתמש בספריות הקוד שיימצאו במחיצה הזאת, עליכם ליידע את המהדר של ג'אווה כי עליו לחפש את הספריות גם במחיצה הזאת.

טעימה מהחומר

בואו ננסה לטעום קצת מהחומר שמצפה לנו בתחילת הסמסטר. נלמד שלושה נושאים מאוד בסיסיים:

1. שימוש בזכרון.
2. התנייה בתכנית.
3. לולאה בתכנית.

שימוש בזכרון

כאמור, המעבד של המחשב משתמש בזכרון על מנת לבצע חישובים מורכבים. גם אנחנו, כמתכנתים, נצטרך לעיתים קרובות להשתמש בזכרון. הדרך הבסיסית בתכנות לשימוש בזכרון היא ע"י מספר פעולות:

1. הקצאת מקום בזכרון.
2. עדכון ערכים במקום שהוקצה.

3. קריאת הערכים הנמצאים שם כרגע.

הקצאת מקום בזכרון

כדי להקצות מקום בזכרון ישנה אבן בניין בתכנות הנקראת "משתנים". משתנה הוא תא בזכרון בו אנו יכולים להשתמש על מנת לאכסן ערכים באופן זמני, לשנות אותם ולקרוא אותם. בכדי שנוכל להשתמש בזכרון, עלינו ראשית להחליט איזה סוג של נתונים אנו מתכוונים לאחסן בו. הסוגים העיקריים הם:

1. ערך מספרי (שלם או לא).
2. טקסט.
3. מצביע למקום אחר בזכרון (מתקדם).

ישנם סוגים רבים של ערכים מספריים, ואת רובם נלמד בקורס. כרגע נתמקד במשתנה מסוג אחד: ערך מספרי שלם. הכוונה היא לכל מספר עשרוני ללא שברים, חיובי, שלילי או אפס. למספר שכזה אנו קוראים Integer. בכדי להקצות מקום בזכרון למשתנה מסוג Integer, עלינו לבחור לו שם, בכדי שנוכל לגשת לאותו המקום בזכרון מאוחר יותר, עפ"י השם (כמה נוח!).

בואו ניצור משתנה בשם myFirstInteger מסוג Integer:

```
int myFirstInteger;
```

שימו לב: זהו מבנה סטנדרטי להקצאת מקום בזכרון עבור משתנה. תמיד נתחיל בציון סוג המשתנה (int במקרה שלנו), אחר כך כל שם שנבחר (ללא רווחים), ולבסוף נקודה-פסיק.

לאחר הרצת השורה הזאת, ג'אווה (או ליתר דיוק ה JVM), תקצה לנו מקום מיוחד בזכרון רק בשבילנו. במקום זה נוכל להשתמש בהמשך התכנית כאוות נפשנו (בהתאם למגבלות השפה כמובן).

עדכון ערכים במקום שהוקצה

בכדי להשתמש בזכרון, עלינו לשים בו ערכים. מאחר שהקצינו מקום לערכים מספריים שלמים (מסוג integer), נוכל לשים במקום הזה רק מספרים כאלה. הדרך לעדכן ערכים במקום שהוקצה היא ע"י שימוש במשתנה שהגדרנו קודם, כך:

```
myFirstInteger = 12;
```

לאחר הרצת השורה הזאת, המקום בזכרון שג'אווה היקצתה עבור המשתנה הזה יכיל את המספר 12.

שימו לב, שאיננו חייבים לקבוע מספר ספציפי. אנו יכולים גם לכתוב ביטוי מתמטי. ג'אווה תחשב את הביטוי, ותשים את התוצאה של הביטוי במשתנה. כך למשל, בשביל לשים את הערך 12 במשתנה, יכולנו לכתוב גם:

```
myFirstInteger = 3 * 4;
```

קריאת ערכים מהזכרון

הרי אין באמת שימוש לאחסון מידע בזכרון מבלי להשתמש בו. הדרך לקרוא מידע מהזכרון היא פשוטה ביותר – כותבים את שם המשתנה במקום בו נדרש נתון. למשל, כזכור לכם, הפקודה System.out.println() מקבלת כפרמטר נתון אשר אותו ג'אווה תדפיס למסך. הנתון הזה יכול להיות מספר או טקסט. מה אם ניתן לפקודה, כפרמטר, את המשתנה שלנו? בואו ננסה:

```
System.out.println(myFirstInteger);
```

בהרצת הפקודה הזאת ג'אווה תדפיס למסך את המספר 12.

שימו לב לתכנית הבאה, וודאו שאתם מבינים היטב מה קורה בכל שורה בה:

```
int myFirstInteger;
int mySecondInteger;
myFirstInteger = 12;
mySecondInteger = myFirstInteger - 2;
System.out.println(mySecondInteger / 2);
```

התנייה בתכנית

עד כה ראינו תכניות ש"רצות" שורה אחרי שורה. נניח שהרצנו מספר פקודות, ולפני שאנו מריצים את הפקודה הבאה, אנו רוצים לבחור אחת משתי פקודות אפשריות. למשל, נניח שיש לנו שני משתנים עם ערכים שונים, ואנו רוצים להדפיס רק את המשתנה בעל הערך הגדול יותר? כלומר, אם myFirstInteger גדול יותר, אזי נדפיס אותו. אחרת נדפיס את mySecondInteger.

בשפות תכנות בכלל ובג'אווה בפרט, באות לעזרתנו פקודות רבות המאפשרות לשלוט בזרימת התכנית. הפקודות נקראות Flow Control Statements. הפקודה אותה אנו מחפשים היא פקודת התנייה, הנקראת, ממש כמו באנגלית, if. בואו ננסה לכתוב את התכנית בעברית, אחר כך נתרגם לאנגלית, ולבסוף לג'אווה:

1. הקצה מקום למשתנה מספרי וקרא לו myFirstInteger
2. הקצה מקום למשתנה מספרי וקרא לו mySecondInteger
3. שים את הערך 12 במשתנה myFirstInteger.
4. שים ב mySecondInteger את תוצאת הביטוי myFirstInteger – 2
5. אם הערך המאוחסן ב myFirstInteger גדול מהערך המאוחסן ב mySecondInteger, אזי
   a. הדפס את הערך המאוחסן ב myFirstInteger.
6. אחרת,
   a. הדפס את הערך המאוחסן ב mySecondInteger.

אתם וודאי שואלים את עצמכם מדוע לשאול איזה ערך גדול יותר, אם ברור שהמשתנה הראשון גדול יותר? התשובה נעוצה בעובדה, שמשתנים, כשמם כן הם, ערכם משתנה במהלך התכנית, ולא תמיד נדע מראש מה ערכו של כל משתנה ואיזה משתנה גדול מהאחר.

שימו לב גם, ששתי שורות ההדפסה נכתבו בשורות נפרדות עם רווחים מתחוילת השורה. זה נקרא "הזחה". הזחה נועדה לוודא שהקוד שלנו קריא ויפה, ומתכנתים אחרים שקוראים אותו יוכלו לקרוא אותו בנקל.

כעת בואו ננסה לתרגם את הטקסט שלנו לאנגלית:

1. Allocate memory for an integer variable and call it myFirstInteger
2. Allocate memory for an integer variable and call it mySecondInteger
3. Assign the variable myFirstInteger with the value 12
4. Assign the variable mySecondInteger with the result of the expression myFirstInteger – 2
5. If the value in myFirstInteger is larger than the value in mySecondInteger, then
   a. Print the value in myFirstInteger
6. Else,
   a. Print the value in mySecondInteger

כעת נותר החלק הקל – לתרגם את התכנית שלנו לשפת ג'אווה.

מדוע זה החלק הקל? מאחר שג'אווה מאוד דומה לאנגלית. שימו לב למילים המודגשות בצהוב. לכל מילה כזאת מבנה משלה בג'אווה. המבנה הזה נקרא Syntax. בואו ננסה לתרגם את התכנית לפי מה שלמדנו עד עכשיו:

```
int myFirstInteger;
int mySecondInteger;
myFirstInteger = 12;
mySecondInteger = myFirstInteger - 2;
if (myFirstInteger > mySecondInteger)
      System.out.println(myFirstInteger);
```

```
else
        System.out.println(mySecondInteger);
```

עיברו על השורות בתכנית, השוו אותן לשורות בטקסט העברי או האנגלי, וודאו שאתם מבינים את מלאכת התרגום. מה יקרה אם תחליפו את סימן המינוס בביטוי המתמטי בסימן הפלוס? מה יודפס אז?

לולאה בתכנית

לולאה היא עוד אחת מ Flow Control Statements. בג'אווה יש לנו מספר סוגים של לולאות, ואנו נלמד כרגע רק סוג אחד. אבל ראשית, ננסה להסביר את הקונספט.

לולאה היא הדרך שלנו, המתכנתים, לומר לג'אווה להריץ פקודה (או קבוצת פקודות) שוב ושוב עד שנרצה להפסיק. כך, למשל, לולאה יכולה להיות:

1. היכנס לחדר המדרגות
2. כל עוד לא הגענו לקומה השלישית
    a. עלה מדרגה אחת
3. הכנס הביתה

איך מתבצע סט הפקודות ("אלגוריתם") הנ"ל? ראשית אנו נכנסים לחדר המדרגות. אחר כך אנו מבצעים בדיקה – האם הגענו לקומה השלישית? אם לא, אנו עולים מדרגה אחת. עכשיו אנו חוזרים לבדיקה – האם הגענו לקומה השלישית? אם לא – אנו עולים עוד מדרגה, וכך הלאה. בשלב כלשהו, כאשר נגיע לבדיקה, נגלה שהגענו לקומה השלישית (למה אין מעלית??). כאשר הבדיקה גילתה שהגענו לקומה השלישית, לא נעלה עוד מדרגה, אלא נמשיך הלאה לפקודות הבאות. הפקודה הבאה אחרי הלולאה היא "הכנס הביתה". אז ניכנס.

בואו נראה מה ה Syntax של פקודת הלולאה בג'אווה:

- while (Boolean expression)
    - statement

שימו לב: Boolean expression הוא ביטוי בג'אווה שהתוצאה שלו אינה מספרית, אלא תוצאה של אמת או שקר, נכון או לא נכון, true או false. כבר נתקלנו בה בתכנית הקודמת בפקודה:

```
if (myFirstInteger > mySecondInteger)
```

הביטוי `myFirstInteger > mySecondInteger` הוא ביטוי בוליאני, מאחר שהתוצאה שלו היא אמת (אם המשתנה הראשון גדול מהשני) או שקר (אם לא). בלולאת while, הפקודה הרשומה במקום statement תתבצע שוב ושוב, כל עוד הביטוי הבוליאני המהווה את תנאי הלולאה הוא אמת. כאשר הביטוי הוא שקר, הלולאה מסתיימת, והתכנית ממשיכה לפקודות הבאות.

שיומ לב לדוגמה הבאה, וודאו שאתם מבינים מה היא מבצעת:

```
while (myFirstInteger > mySecondInteger)
        mySecondInteger = mySecondInteger + 1;
```

חישבו היטב על כל חלק בביטוי. נסו להבין מתי הוא מסתיים. מה יקרה בלולאה הבאה:

```
while (myFirstInteger > mySecondInteger)
        mySecondInteger = mySecondInteger - 1;
```

הלולאה לא תסתיים לעולם! הביטוי הבוליאני המהווה את תנאי הלולאה לעולם לא יהיה שקר! ראשית, וודאו שאתם מבינים מדוע.

לולאה שאינה מסתיימת לעולם, כלומר, שתנאי הלולאה לעולם אינו שקר, נקראת "לולאה אינסופית" או "infinite loop".
לולאה אינסופית היא בעיה ידועה בתכנות. כך, למשל, כאשר "נתקעת" לכם תוכנה במחשב, סיכוי טוב שמדובר בלולאה אינסופית שאינה מאפשרת לתכנית להתקדם.

קיבוץ פקודות

לפעמים נרצה יותר מפקודה אחת בתוך הלולאה. בכדי לקבץ מספר פקודות תחת קורת גג אחת, נתחום אותן בסוגריים מסולסלים. כך, למשל, הלולאה הבאה תדפיס את השורה "hello" 5 פעמים:

```
int count = 0;
while (count < 5) {
      System.out.println("hello");
      count = count + 1;
}
```

השימוש בסוגריים מסולסלים נפוץ מאוד בתכנות, והוא מאפשר לנו להשתמש במספר פקודות במקומות המאפשרים הרצה של פקודה אחת. כך, למשל, ניתן להשתמש בסוגריים מסולסלים בפקודת if, אותה הכרנו בחלק הקודם.

תרגיל מסכם

כיתבו תוכנית המקצה מקום למשתנה (ביחרו אתם את שמו), שמה בו ערך, ומדפיסה למסך את הערך המוחלט שלו.
לאחר מכן, הדפיסו למסך כוכביות. מספר הכוכביות יהיה שווה לערך המוחלט של המספר.

בהצלחה!

עזרה

אתר הקורס כולל מידע חיוני על הקורס אך גם משאבי מידע בנושא תיכנות ג'אווה:

http://www.intro2cs.com

הוראות הכנת סביבת העבודה ועוד חומר קריאה לקראת תחילת הסמסטר:

http://www.intro2cs.com/working-environment.html

לימוד ג'אווה צעד אחר צעד. מצוין למי שרוצה להקדים וללמוד את החומר לקראת השיעור, או למי שפתח פער:

http://java.sun.com/docs/books/tutorial/java/

ספר ייעוץ/עיון לגבי כל פקודות ג'אווה האפשריות:

http://java.sun.com/javase/6/docs/api/