תוכנית ה Java הראשונה שלי

על מנת לבצע תרגול זה, עליכם להתקין JDK על פי ההנחיות באתר. שימו לב, גירסת ה Java שבה נשתמש בקורס היא Java 8 (תת הגירסא לא משנה). במידה ותעבדו עם eclipse, אתם יכולים לעבוד עם המדריך הבא: eclipse

http://courses.cs.tau.ac.il/software1/2021a/misc/workenv.pdf

נכלל בתרגול:
כתיבת תוכנית ראשונה.
טיפוסים בסיסיים ב Java (פרימיטיביים + מחרוזות).
המרה של טיפוסים פרימיטיביים.
פעולות בסיסיות על טיפוסים פרימיטיביים.

חלק א' – תוכנית ראשונה:
התוכנית הראשונה שנכתוב היא תוכנית אשר מדפיסה את המחרוזת !Hello World, והיא נראית כך:

```
public class HelloWorld {
	public static void main(String[] args) {
		System.out.println("Hello World!");
	}
}
```

שימו לב לדברים הבאים:

1. שם התוכנית הוא HellowWorld, והיא נכתבת במחלקה (class) שנקראת HelloWorld וכתובה בקובץ שנקרא HelloWorld.java. ב Java כל הקוד נכתב במחלקות, לא ניתן לכתוב פונקציות\להגדיר משתנים בתוך קבצים מחוץ למחלקה.
2. בתוך המחלקה HelloWorld יש פונקציה אחת שנקראת main. הפונקציה מקבלת מערך של מחרוזות ולא מחזירה שום ערך.
   a. המילה void המופיעה בחתימת הפונקציה מציינת את סוג ערך ההחזרה שלה. ב Java עלינו להגדיר את סוג ההחזרה של כל פונקציה.
   b. המשמעות של String[] args היא: הפונקציה מקבלת פרמטר יחיד שנקרא args. לכל פרמטר עלינו להצמיד טיפוס. במקרה זה, הטיפוס הוא **מערך** של מחרוזות (String). בשלב הזה ניתן להניח שמערך הוא סוג של רשימה, טיפוס שאתם מכירים משפת ה Python, אבל מערך הוא לא בדיוק רשימה כיוון שיש כמה תכונות של רשימה שלא קיימות במערכים – למשל, לא ניתן להוסיף או למחוק איברים מהמערך.
   c. בשונה מ Python, לא ניתן להגדיר משתנה מבלי להגדיר את הטיפוס שלו, לכן עלינו לציין את הטיפוס של args.
3. ב Java אין משמעות תחבירית לרווחים וירידות שורה. פתיחת וסגירת בלוק נעשית באמצעות סוגריים מסולסלים. במילים אחרות – התוכנית HellowWorld יכלה להיכתב בשורה אחת, וירידות השורה וההזחות נעשות עבור הקריאות של הקוד.
4. ההדפסה נעשית באמצעות קריאה לפונקציה System.out.println.
5. בסוף כל פקודה ב Java עלינו לשים את התו ; (נקודה פסיק)

כעת, הריצו את התוכנית ב Eclipse וודאו שאתם רואים את הפלט שלה בחלונית ה Console.
מה יקרה אם נשנה את שמה של הפונקציה main, או שנשנה את ערך ההחזרה שלה\נוריד פרמטר? התשובה היא שבמקרה זה התוכנית לא תרוץ. על מנת להריץ תוכנית X ב Java, המחלקה X צריכה לכלול פונקציית main שחתימתה זהה לזו שהצגנו כאן, אחרת לא ניתן יהיה להריץ אותה כתוכנית.

חלק ב' – הגדרת משתנים:

נרחיב את התוכנית שלנו לעבודה עם משתנים, ועל הדרך נכיר טיפוסים נוספים בשפת Java.

```
public class Example1 {
	//one line comment
	public static void main(String[] args) {
		/*
		 multi-line comment
		 */

		int i;	//*
		int j = 1;
		boolean bool = false;
		double d = 1.56;
		char c = 'a';
		String s = "a";
		i = 5;	//**
		System.out.println("(1): " + i + " " + s);
		System.out.println("(2): " + bool);
		System.out.println("(3): " + d);
		System.out.println("(4): " + c);
		System.out.println("(5): " + c + j);
		System.out.println("(6): " + (c + j));
		System.out.println("(7): " + (char)(c + j));
	}
}
```

בתוכנית הבאה ניתן לראות כמה דברים חדשים:

1. שני סוגי הערות שניתן לכתוב בתוך הקוד.
2. הגדרת משתנים בתוך פונקציה. בפונקציה main יש שבעה משתנים:
   a. המשתנה args שהוא הפרמטר לפונקציה, וכבר הכרנו אותו בחלק הקודם.
   b. המשתנה i שטיפוסו הוא int (קיצור של integer, מספר שלם). בשורה שבה i מוגדר (השורה המסומנת ב *) לא בוצעה עדין השמה ל i ולכן לא ניתן יהיה להשתמש בו עד לביצוע השורה שבה מתבצעת ההשמה (השורה המסומנת ב **). אם ננסה למשל להדפיס את i לפני ביצוע ההשמה נקבל שגיאת קומפילציה (שגיאה תחבירית).
   בנוסף לטיפוס int, קיימים עוד טיפוסים המייצגים מספרים שלמים. הם נבדלים ביניהם בכמות המקום שהם תופסים בזכרון, ובהתאם לזה גם בטווח הערכים שהם יכולים לייצר (נראה בהמשך).
   c. המשתנה j גם הוא מטיפוס int, והוא מקבל את ההשמה שלו באותה שורת הקוד שבה הוא מוגדר.
   d. המשתנה bool הוא מטיפוס boolean. בדומה לטיפוס הבוליאני שאתם מכירים מ Python הוא יכול לקבל שני ערכים אפשריים – true/false (להבדיל מפייתון, שני הערכים פה מתחילים באות קטנה).
   e. המשתנה d מטיפוס מספר עשרוני. ב Java יש שני טיפוסים שונים עבור יצוגים של מספרים עשרוניים, double ו float הנבדלים בכמות המקום שבהם תופסים בזכרון.
   f. המשתנה c מטיפוס char, שהוא קיצור של character.

הטיפוס character אינו מוכר לכם מפייתון. טיפוס זה מייצג תו בודד שיכתב בתוך גרשיים. לצורך העניין, גם תו ירידת השורה \n נחשב לתו בודד. שימו לב להבדל בין המשתנה c למשתנה s. המשתנה s הוא מטיפוס מחרוזת, והערך שלו נכתב בין מרכאות (ב Java מחרוזות נכתבות בין מרכאות ותוים נכתבים בין גרשיים). בין המרכאות ניתן לכתוב כמה תוים שרוצים, כולל מחרוזת ריקה.

אז מהו בעצם ההבדל בין s לבין c? הטיפוסים שלהם שונים, ולכן ניתן לבצע עליהם פעולות שונות, בדיוק כשם שעל רשימה בפייתון ניתן לבצע פעולות שונות מאלה שניתן לבצע על מספר שלם.
מלבד ההבדל בפעולות, יש גם הבדל בערכים שכל אחד מהמשתנים יכול לקבל. המשתנה s יכול לקבל השמה לערך "abc", אבל לא ניתן לבצע השמה דומה למשתנה c. אם ננסה לבצע השמה ל abc נקבל שגיאה על כך שלא ניתן לבצע השמה של מחרוזת לתוך משתנה מטיפוס character. אם ננסה לבצע השמה ל 'abc' (כלומר, גרשיים במקום מרכאות), נקבל שגיאה, כיוון שבין מרכאות אמור להופיע תו יחיד.

כעת, הריצו את הקוד והתבוננו בפלט שלו:
שורה (1) – ב Java ניתן לשרשר מחרוזות עם טיפוסים שאינם מחרוזות, כך שבשורה זו אין בעיה לשרשר את i שהוא int למחרוזות אחרות. באופן דומה מבוצעים שרשורים כאלה בשורות 2-4.
תוצאת השרשור היא מחרוזת אחת שבסופו של דבר מודפסת למסך ע"י הפקודה System.out.println.

שורות (5) ו (6) – שורה (5) מתנהגת "כצפוי" – אנחנו מבצעים פה שרשור של מחרוזת, תו ומספר שלם ומקבלים מחרוזת אחת המכילה את כולם. מכיוון שיש פה שלוש פעולות שרשור, קודם משורשרת המחרוזת ":(5)" לתו c, מתקבלת מחרוזת תוצאה, ואליה משורשר המשתנה j.
בשורה (6) סדר הפעולות שונה. קודם כל אנחנו מבצעים פעולת חיבור בין c לבין j, ורק אחר כך מבצעים את השרשור למחרוזת. על מנת להבין מהי תוצאת החיבור בין c לבין j אנחנו קודם כל צריכים לחשוב על הטיפוס של התוצאה המתקבלת. מדובר בפעולת חיבור בין תו למספר שלם, כך שלא ניראה סביר שהתוצאה צריכה להיות מחרוזת. הגיוני שהתוצאה תהיה מספר שלם או תו. או שבעצם אין הרבה הבדל?
ב Java (וגם בשפות תכנות אחרות כמובן) ניתן להמיר כל תו למספר שלם, בהתאם לטבלת הקידוד בה משתמשים בשביל לקודד את התוים. לדוגמא, אפשר לקודד תוים לפי קידוד ascii או לפי קידוד Unicode. כלומר, אנחנו רואים שבעצם לכל תו מתאים מספר שלם יחיד, וניתן לעשות מעבר מהיר בין תוים למספרים שלמים.
לכן, בבואנו לחבר בין c לבין j אנחנו בעצם מבצעים חיבור של שני מספרים שלמים. התוצאה היא מספר שלם, והוא גם זה שיודפס בשורה (6).
שורה (7) – שורה זו נבדלת משורה (6) בכך שתוצאת החיבור בין c ל j עוברת המרה (casting) לטיפוס character. כלומר, אנחנו לוקחים את הערך המספרי שמתקבל וממירים אותו לתו (character). לכן, בשורה זו נקבל הדפסה של התו שמתאים לערך המספרי שהודפס בשורה 6.
בחלק ג' של תרגול זה נחזור לדבר על פעולת ה casting ומשמעותה.

<u>חלק ג' – הטיפוסים הפרימיטיביים:</u>
כעת, ננסה להתעמק בטיפוסים השונים שראינו.
ב Java יש 8 סוגי טיפוסים בשפה שנקראים "פרימיטיביים", ושאר הטיפוסים נקראים "אובייקטים" או "רפרנסים" (references).
מספרים שלמים:
מיוצגים ע"י ארבעה טיפוסים: byte, short, int, long
מספרים עשרוניים מיוצגים ע"י שני טיפוסים: float, double
ערכים בוליאניים: Boolean
תו: character
כל אחד מהטיפוסים הפרימיטיביים תופס כמות קבועה של מקום בזכרון. למשל, int יתפוס 4 בתים (32 bit). לכן, טווח הערכים האפשריים עבור int הוא תחום בין שני קצוות.

Primitive Types					
Type Name	Wrapper class	Value	Range	Size	Default Value
byte	java.lang.Byte	integer	−128 through +127	8-bit (1-byte)	0
short	java.lang.Short	integer	−32,768 through +32,767	16-bit (2-byte)	0
int	java.lang.Integer	integer	−2,147,483,648 through +2,147,483,647	32-bit (4-byte)	0
long	java.lang.Long	integer	−9,223,372,036,854,775,808 through +9,223,372,036,854,775,807	64-bit (8-byte)	0
float	java.lang.Float	floating point number	±1.401298E−45 through ±3.402823E+38	32-bit (4-byte)	0.0
double	java.lang.Double	floating point number	±4.94065645841246E−324 through ±1.79769313486232E+308	64-bit (8-byte)	0.0
boolean	java.lang.Boolean	Boolean	true or false	8-bit (1-byte)	false
char	java.lang.Character	UTF-16 code unit (BMP character or a part of a surrogate pair)	'\u0000' through '\uFFFF'	16-bit (2-byte)	'\u0000'

(הטבלה לקוחה מגירסא מוקדמת של
https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types)

לרוב מקובל להשתמש ב int בשביל לייצג מספרים שלמים, ו float בשביל לייצג מספרים עשרונים, אבל כדאי להכיר את הטיפוסים השונים על מנת לנהל את הזיכרון בצורה האופטימלית.
כאשר אנחנו מגדירים משתנה כ int i, בזיכרון התוכנית מוקצים 4 בתים עבור ערכו של i. כל השמה של i לערך שלם תבצע כתיבה לאותו האזור בזיכרון. לכן, אם אנחנו יודעים שערכו של i הולך להיות מספר יחסית קטן (למשל, i מציין מספר סטודנטים הרשום לקורס), ניתן בהחלט להשתמש בטיפוס short או אפילו byte.
מה היתרון של קביעת הטיפוס של משתנה מבעוד מועד?

א. יעילות בזמן חישובים – למשל: חישובים עם שלמים הם מהירים יותר מאשר חישובים על מספרים עשרוניים, ולכן משתלם לנו להגדיר מראש שמשתנה מסויים יהיה מספר שלם.
ב. חסכון בהקצאת זכרון – כפי שאמרנו, אפשר להשתמש מראש בטיפוסים שתופסים פחות מקום בזיכרון אם ידוע לנו טווח הערכים.
ג. הטיפוס של נתון מסויים מכתיב את הפעולות שניתן לבצע עליו. למשל, על שלמים ניתן לבצע פעולות אריתמטיות. על טיפוסים שהם boolean ניתן לבצע פעולות בולאניות, וכו'.

ומה עם מחרוזות? הטיפוס String אינו טיפוס פרימיטיבי, ושייך לקטגוריה השניה שנקראת "אובייקט".

האם ניתן לבצע המרות בין הטיפוסים הפרימיטיביים השונים? למשל, היינו רוצים להמיר int ל short ולהיפך. אינטואיטיבית, אין שום בעיה להמיר short ל int. אם ניקח מספר שלם שתופס 2 בתים בזיכרון, לא אמורה להיות שום בעיה לכתוב אותו לתוך משתנה שאמור לייצג מספר שלם שתופס 4 בתים בזיכרון. ואכן, ההשמה הזו עובדת בצורה חלקה:

```
int i = 2400000;
short s = 2;
i = s;
```

מה יקרה אם ננסה לבצע השמה הפוכה, מ i ל s? המקרה הזה הוא בעייתי יותר, כיוון שאנחנו רוצים להעתיק תוכן שנשמר בתוך 4 בתים לתוך משתנה שעבורו מוקצים רק שני בתים. במקרה שלנו, i מכיל ערך שגדול מדי מטווח הערכים של short, ולכן ההשמה מ i ל s תגרום ל"אבוד מידע" – כלומר, לא נוכל לכתוב את כל 4 הבתים של i לתוך s. לעומת זאת, אם i היה שווה ל 12, זהו ערך שיכול להיכתב בשלמותו בתוך s ולכן בהשמה מ i ל s לא אמור להיות איבוד מידע.
מכיוון שהשמה מ int ל short היא לא בטוחה, ב Java לא ניתן לבצע את ההשמה בפשטות כמו שראינו בדוגמא ההפוכה. אם ננסה לכתוב s =i ב eclipse נקבל שגיאה. השגיאה שנקבל היא שגיאה תחבירית – בשפת ה Java זה לא חוקי תחבירית לבצע השמה מ int ל short. בשביל בכל זאת לבצע את ההשמה הזו, אנחנו צריכים להשתמש בפעולה שנקראת casting – המרה.

```
int i = 2400000;
short s = 12;
s = (short)i; //casting
System.out.println(s);
```

מה יודפס כשנדפיס את s? הריצו את הקוד (צריך להעתיק אותו לתוך תוכנית עם פונק' main כדי שירוץ) ובדקו.

אותו העיקרון של המרות בטוחות ולא בטוחות מתקיים כמובן בין עוד זוגות של טיפוסים. למשל, בין byte ל long ובין float ל double.
לקריאה נוספת על המרות בטוחות ולא בטוחות:
https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html

חלק ד (פעולות מתקדמות על טיפוסים פרימיטיביים):
הריצו את התוכנית הבאה:

```
public class Example2 {
	public static void main(String[] args) {
		int num1 = 15;
		boolean b1 = func1(num1);
		boolean b2 = func2(num1);
		System.out.println("and :" + (b1 && b2));
		System.out.println("or :" + (b1 || b2));
		System.out.println("------------------");
		System.out.println("-- calculating b3: ");
		boolean b3 = func1(num1) && func2(num1);  //*
		System.out.println("-- calculating b4: ");
		boolean b4 = func1(num1) || func2(num1); //**
		System.out.println("not :" + !b4);
	}

	public static boolean func1(int x) {
		System.out.println("func1");
		return x > 10;
	}

	public static boolean func2(int x) {
		System.out.println("func2");
		return x % 2 == 0;  //% is modulo, same as in Python
	}
}
```

הפונקציות func1 ו func2 הן פונקציות שמחזירות ערך בוליאני, והפונקציה main קוראת להן. כשנריץ את התוכנית Example2, הפונקציה שתופעל היא כמובן הפונקציה main.

בתוכנית זו ניתן לראות:

א. שתי ההדפסות הראשונות מדגימות כיצד כותבים ב Java את פעולות ה and (וגם) ו or (או) ב Java. פעולות אלה, כזכור, מבוצעות על ערכים בוליאנים ומחזירות ערך בוליאני.

ב. בשורות המסומנות ב * וב ** מתבצעות כן גם פעולות של and ו or, אך ההבדל הוא שהן מבוצעות על תוצאות הקריאה לפונקציות func1 ו func2, ודרך הדוגמא הזו ניראה תכונה של && ו || שהיא לא אינטואיטיבית. התבוננו בהדפסות שמתבצעות בשורה **. אנחנו רואים שמתבצעת רק ההדפסה עבור func1 אך לא עבור func2. מדוע? כיוון שהקריאה ל func1 מחזירה true ולכן ברור שלא משנה מה

הערך בצידו הימני של האופרטור ||, התוצאה תהיה true. לכן, הוא כלל לא מחושב ולא מבוצעת קריאה ל func2.

ג. בשורה האחרונה של התוכנית ניתן לראות כיצד כותבים את פעולת ה not.

כעת הריצו את התוכנית Example3

```
public class Example3 {
	public static void main(String[] args) {
		int num1 = 15;
		int num2 = num1++;
		System.out.println("num1 is " + num1);
		System.out.println("num2 is " + num2);
		int num3 = 12;
		int num4 = ++num3;
		System.out.println("num3 is " + num3);
		System.out.println("num4 is " + num4);

	}
}
```

בתוכנית Example3 אנו רואים רואים את האופרטור ++ בשתי צורות.
האופרטור ++ מבצע שתי פעולות בו זמנית: הוא גם מגדיל ב 1 את ערכו של המספר עליו הוא מופעל, וגם מחזיר את המספר עצמו. מה קורה קודם? הקידום או ההחזרה? הריצו את הקוד והסיקו כיצד כל אחת מאופציות ההפעלה עובדת. בדוגמא ל ++ ניתן להשתמש גם ב --.