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 21 (תת הגירסה לא משנה). אם תעבדו עם Eclipse תוכלו להיעזר במדריך הבא:

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

מה נכלל בעבודה עצמית זו?

- ✓ כתיבת תוכנית ראשונה
- ✓ טיפוסים בסיסיים ב Java
- ✓ 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 HelloWorld, והיא נכתבת במחלקה (class) שנקראת HelloWorld וכתובה בקובץ HelloWorld.java. ב Javaכל הקוד נכתב במחלקות, לא ניתן לכתוב פונקציות או להגדיר משתנים מחוץ למחלקה.
2. בתוך המחלקv HelloWorld יש פונקציה אחת שנקראת main. הפונקציה מקבלת מערך של מחרוזות ולא מחזירה שום ערך.
   a. המילה void מציינת את סוג ערך ההחזרה של הפונקציה. יש להגדיר את סוג ע רך ההחזרה של כל פונקציה.
   b. המשמעות של String []args היא: הפונקציה מקבלת פרמטר יחיד שנקרא args. לכל פרמטר יש להצמיד טיפוה. במקרה זה, הטיפוס הוא מערך של מחרוזות. יש דמיון חלקי בין מערך לרשימה (טיפוס שהכרתם ב Python), אך לרשימות יש תכונות שלא קיימות במערכים. למשל, לא ניתן להוסיף או למחוק איברים ממערך.
   c. עלינו לציין את הטיפוס של args, כי ב Java(בשונה מ Pyhton) לא ניתן להגדיר משתנה מבלי לציין את הטיפוס שלו.
3. ב Java אין משמעות תחבירית לרווחים וירידות שורה. פתיחת וסגירת בלוק נעשית באמצעות סוגריים מסולסלים. ירידות השורה וההזחות נעשות רק למען שיפור קריאות 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 של/ נוריד פרמטר?
התשובה היא שבמקרה כזה 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 זו ניתן לראות:

1. שני סוגי הערוץ שניתן לכתוב בקוד.
2. הגדרת משתנים בתוך פונקציה. בפונקציה main בתוכנית הנ"ל יש שבעה משתנים:
   a. המשתנה args שהוא הפרמטר לפונקציה.
   b. המשתנה i שטיפוסו הוא int. בשורה שבה i מוגדר (השורה המסומנת ב *) לא בוצעה עדיין השמה ל i ולכן לא ניתן יהיה להשתמש בו עד לביצוע השורה שמסומנת ב ** - בה מתבצעת ההשמה. אם ננסה למשל להד]יס את i לפני ביצוע ההשמה נקבל שגיאת קומפילציה.
   בנוסף לטיפוס int, קיימים עוד טיפוסים המייצגים מספרים שלמים. הם נבדלים זה מזה בכמות המקום שהם תופסים בזכרון, ובהתאם לזה גם בטווח הערכים שהם יכולים לייצר (פרטים בהמשך).
   c. המשתנה j גם הוא מטיפוס int, מקבל את ההשמה שלו בשורת הקוד שבה הוא מוגדר.
   d. המשתנה bool מטיפוס boolean. יכול לקבל שני ערכים אפשריים true/false (בניגוד לפייתון כותבים את הערכים עם אותיות קטנות בלבד).
   e. המשתנה d מטיפוס נקודה צפה. ב Java יש שני טיפוסים שונים עבור יצוגים של מספרי נקודה צפה: float ו double, שנבדלים זה זה בכמות המקום שהם תופסים בזכרון.
   f. המשתנה c מטיפוס char, שהוא קיצור של character.
   g. המשתנה s מטיפוס מחרוזת. ערכו נכתב בין מרכאות.

הטיפוס char אינו מוכר לכם מפייתון. טיפוס זה מייצג תו בודד שייכתב בתוך גרשיים. (גם תו ירידת השורה \n נחשב תו בודד.) שימו לב להבדל בין המשתנה c למשתנה s. ב Java מחרוזות נכתבות בין מרכאות ותווים בין גרשיים. בין המרכאות ניתן לכתוב רצף של 0 תווים או יותר ("" היא מחרוזת ריקה).
הטיפוסים של s ו c שונים ולכן ניתן לבצע עליהם פעולות שונות, בדיוק כשם שעל רשימה בפייתון ניתן לבצע פעוות שונות מאלה שניתן לבצע על מספר שלם.
בנוסף, ישנו גם הבדל בערכים שהמשתנים s ו c יכולים לקבל. המשתנה s יכול לקבל השמה לערך "abc" אבל לא ניתן לבצע השמה דומה למשתנה c. (אם ננסה לבצע השמה ל 'abc' – עם גרשיים במקום מרכאות – לתוך c נקבל שגיאה כי בין גרשיים אמור להופיע תו יחיד.

כעת, הריצו את הקוד והתבוננו בפלט שלו:
שורה (1) – ב Java ניתן לשרשר מחרוזות עם טיפוסים שאינן מחרוזות (i שהוא int משורשר למחרוזת).
שירשורים כאלה מתבצעים בשורות 2-4. תוצאת השרשור היא מחרוזת אח שבסופו של דבר מודפסת למסך ע"י הפקודה System.out.println.

בשורה (5) יש כמה פעולות שירשור. קודם משורשרת המחרוזת ":(5)" לתו c, מתקבלת מחרוזת תוצאה, ואלי משורשר המתשתנה j.
בשורה (6) סדר הפעולות שונה. קודם מתבצע חיבור בין c ו j (חיבור בין תו למספר שלם). מה תוצאת החיבור הזה?
ב Java, כמו בשפות רבות, ניתן להמיר כל תו למספר שלם בהתאם לטבלת הקידוד בה משתמשים לקודד את התווים (לדוגמה, קידוד ascii או Unicode). כל תו מתאים למספר שלם יחיד. לכן, החיבור של c ו j מתורגם לחיבור של שני מספרים שלמים. התוצאה היא מספר שלם שיודפס בשורה (6).
שורה (7) – שם תוצאת החיבור של c ו j עוברת המרה (casting) לטיפוס char. לכן בשורה זו תתקבל הדפס של התו המתאים לערך המספרי שהודפס בשורה הקודמת. בחלק הבא של התרגול הזה נדבר על פעולת ה casting ומשמעותה.

<u>חלק ג – הטיפוסים הפרימיטיביים/ היסודיים:</u>

ב Java יש 8 סוגי טיפוסים בשפה שנקראים "פרימיטיביים" (יסודיים). שאר הטיפוסים נקראים רפרנסים (references) או אובייקטים.

יש 4 טיפוסים שמייצגים מספרים שלמים: byte, short, int, long

יש 2 טיפוסים שמייצגים מספרי נקודה צפה: float, double

טיפוס שמייצג ערכים בוליאניים: boolean

טיפוס שמייצג תו: char

כל אחד מהטיפוסים הנ"ל תופס כמות קבועה של מקום בזיכרון, ולכן טווח הערכים של כל טיפוס מוגבל.

Type	Contains	Default value	Size	Range
boolean	true / false	false	1 bit	
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
byte	Signed integer	0	8 bits	-128 to 127
short	Signed integer	0	16 bits	-32768 to 32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	1.4E-45 to 3.4028235E+38
double	IEEE 754 floating point	0.0	64 bits	4.9E-324 to 1.7976931348623157E+308

כאשר אנו מגדירים משתנה i כ Int, בזיכרון התוכנית מוקצים 4 בתים עבור ערכו של i. כל השמה של ערך לתוך i תבצע כתיבה לאותו האזור בזיכרון. לכן, אם יודעים שערכו של i יהיה מספר קטן יחסית ניתן 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 צאשר חישובים על מספרי נקודה צפה)
ב. חסכון בהקצאת זיכרון (אם ידוע מראש טווח הערכים אז אפשר לבחור בטיפוס שתופס פחות מקום בזיכרון)
ג. הטיפוס של משתנה מסויים מכתיב את הפעולות שניתן לבצע עליו (למשל: על שלמים אפשר לסצע פעולות אריתמטיות, על Boolean פעולות בולאניות וכו')

מה לגבי מחרוזות?

הטיפוס String אינו פרימיטיבי ושייך לסוג השני של טיפוסים: טיפוסי רפרנס או אובייקטים.

האם ניתן לבצע המרות בין טיפוסים פרימיטיביים? למשל המרה של int ל short ולהיפך.

אין בעיה להמיר short ל int כי בעצם ניקח מספר שתופס 2 בתים בזיכרון ונכתוב אותו לתוך משתנה שתופס 4 בתים בזיכרון. דוגמה:

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

ומה לגבי השמה הפוכה? זה בעייתי! אנו רוצים להעתיק תוכן שנשמר בתוך 4 בתים לתוך משתנה לו מוקצים 2 בתים בלבד. מידע עלול "ללכת לאיבוד". לכן ב Java לא ניתן לבצע השמה כזו בפשטות. כלומר, אם ננסה לכתוב s=i; נקבל שגיאה (שגיאה תחבירית, כי התחביר של Java לא מאפשר לעשות השמה כזו כך). אם בכל זאת נרצה לבצע השמה כזו נצטרך להשתמש בפעולה שנקראת casting – המרה.

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

בידקו מה יודפס כשנריץ את הקוד הנ"ל. יש להעתיק אותו לתוך תוכנית עם פונ' main.

המרות בטוחות ולא בטוחות ניתן לבצע בין זוגות נוספים של טיפוסים פרימיטיביים. מידע נוסף:

https://docs.oracle.com/javase/specs/jls/se21/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
	}
}
```

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

בתוכנית זו ניתן לראות:

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

ב. בשורה ** רואים שמתבצעת רק ההדפסה עבור התוצאה של func1 אך לא עבור func2. הסיבה היא שהקריאה ל func1 החזירה true ולכן לא משנה מה הערך בצידו הימני של האופרטור || התוצאה תהיה true. לכן ערך זה כלל לא מחושב. התנהגות מקבילהקורית גם באופרטור && כשהערך משמאל לאופרטור הוא false.

ג. בשורה האחרונה של התוכנית מודגם שימוש באופרטור not (מיוצג ע"י !).

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 זו רואים את האופרטור ++ בשתי צורות. אופרטור זה מבצע שתי פעולות: מגדיל ב 1 את ערכו של המספר עליו הוא מופעל, ומחזיר את המספר עצמו. מה קורה קודם? הריצו את הקוד והסיקו מה ההתנהגות עבור כל רחת מאפשרויות ההפעלה.