<u>מבני בקרה</u>

<u>חלק א' – מבנה if/else ואופרטור תנאי</u>

כמו ב Python, גם ב Java ניתן להשתמש במשפטי if else.

לדוגמא:

```
public class IfElse {
	public static void main(String[] args) {
		if (args.length < 1) {
			System.out.println("missing arguments!");
		}
		else {
			int num = Integer.parseInt(args[0]);
			if (num % 2 == 0) {
				System.out.println("*");
			}
			else {   //###

				if (num % 5 == 0) {
					System.out.println("**");
				}
				else {
					System.out.println("***");
				}
			}    //###
		}
	}
}
```

כמה דגשים לבלוק if/else:

א. כל מה שנמצא בתוך בלוק if או else אמור להופיע בסוגריים משולשים.

ב. אם בתוך if/else יש פקודה אחת בלבד, ניתן לוותר על סוגריים משולשים. לדוגמא, השורה שמדפיסה ** יכלה להופיע בתוך ה if ללא סוגרים משולשים, נסו את זה.
למרות שזה עובד, זה לא מומלץ! (מה יקרה אם נוסיף שורות בעתיד?) עדיף להשתמש תמיד בסוגריים משולשים.

ג. אין דרך כזה elif ב Java. במקום זאת, יש לפתוח בלוק if בתוך else (כמו שאנחנו רואים בבלוק ה if אשר מדפיס **.

הריצו את התוכנית עם 5 קונפיגורציות:

1. ללא ארגומנטים בשורת הפקודה.
2. ארגומנט יחיד: 2
3. ארגומנט יחיד: 10
4. ארגומנט יחיד: 15
5. ארגומנט יחיד: 9

ודאו שהפלט ברור לכם.

כעת, מחקו את שתי השורות המסומנות ב ### (למעשה, אנחנו מסירים את בלוק ה else ומוציאים את התוכן שלו רמה אחת למעלה. הריצו שוב את קונפיגורציות 1-5 וודאו שאתם מבינים היכן אמור להיות שינוי, ומדוע.

אופרטור התנאי (נקרא גם אופטרטור טרנרי – ternary operator) מאפשר לבחור חישוב ערך מבין שני חישובים אפשריים.

בפייתון קיים התחביר הבא:

```
x = 1 if y == 0 else 8
```

כלומר, אם y ==0 אז x הוא 1, ואחרת 8

ב Java נבטא את אותו הדבר באמצעות התחביר הבא:

```
int x = y == 0 ? 1 : 8
```

את הביטוי נחלק לשלושה חלקים:

ביטוי בוליאני (צבוע בצהוב).
הערך שמתקבל אם הביטוי הבוליאני הוא True (צבוע בירוק)
הערך שמתקבל אם הביטוי הבוליאני הוא False (צבוע בתכלת)

חלק ב' – מבנה switch/case:

במקרה של ריבוי תנאים, קיים תחביר מיוחד ב Java שנקרא בלוק switch/case.
הריצו את הקוד הבא:

```
public class SwitchCase {
	public static void main(String[] args) {
		int grade = 80;
		switch (grade) {
		case 100:
			System.out.println("A+");
			break;  //*
		case 90:
			System.out.println("A");
			break;  //*
		case 80:
			System.out.println("B");
			break;
		case 70:
			System.out.println("C");
			break;
		case 60:
			System.out.println("D");
			break;
		default:
			System.out.println("F");
			break;
		}
	}
}
```

בלוק ה switch מקבל משתנה מטיפוס שמשתייך לקבוצה הבא: {byte, short, int, char, String}. בהמשך ניראה גם טיפוסי מניה (Enum) יכולים להשתתף בבלוק switch/case.
ה case שיבוצע הוא ה case שתואם לערך שעליו מבוצע ה switch (במקרה שלנו, זהו המשתנה grade).
הבלוק default מטפל בערכים שאין להם case מתאים.
שימו לב שבלוק case/default הוא בלוק מיוחד שבתוכו ניתן לכתוב יותר משורת קוד אחת ללא סוגריים מסולסלים.
הריצו את הקוד על הערכים grade = 100, grade=89, grade=30, grade=90. וודאו שאתם מבינים את הפלט.

כעת, נבצע שני שינויים:

1. הסירו את בלוק ה default (סה"כ 3 שורות קוד). עבור אילו קלטים ריצת התוכנית תיתן פלט שונה? הריצו את התוכנית על אותם הקלטים.
2. החזירו את בלוק ה default והסירו את שתי השורות המסומנות ב *. בדקו את הקוד עבור grade = 100 ו grade = 90.

הסיבה להתנהגות ה"מוזרה" היא זו: ברגע שנכנסים לאחד ה case-ים, מבצעים את כל הפקודות שמופיעות ב case זה וכל ה case-ים העוקבים לו, עד שהם נגמרים או עד שפוגשים את הפקודה break.
לכן, ברוב המוחלט של בלוקי switch/case, תמיד יופיע break בסוף כל case.

חלק ג – לולאת while ו do-while

לולאת ה while עובדת בדומה למה שאתם מכירים מ Python.

תחילה, נבדק תנאי ריצת הלולאה (תנאי בוליאני). במידה והוא מתקיים, גוף הלולאה מבוצע. בתוכו מבוצע גם קידום משתנה העזר שבו תלוי התנאי הבוליאני. הקידום נעשה בשאיפה לעצור את הלולאה בשלב מסויים.
אם בגוף הלולאה לא מבוצע שום קידום של משתנה העזר לכיוון עצירה ואין פסוקית break, הרי שנקבל לולאה אינסופית.

```
public class While {
	public static void main(String[] args) {
		int cnt = 5;
		while (cnt > 0) {
			System.out.println(cnt);
			cnt--;
		}

	}
}
```

הריצו את הקוד הבא וודאו שאתם מבינים מה אמור להיות מודפס. כעת, שנו את ערכו של cnt ל 0 והריצו אותה מחדש.
האם ניתן לצמצם את גוף הלולאה לשורה אחת במקום שתי שורות?
שימו לב שבלולאת while יכול לקרות מצב שלא תתבצע אף איטרציה.
אם אנחנו רוצים לוודא שבכל מקרה, לא משנה מה ערכו של התנאי הבוליאני, איטרציה אחת תתבצע, עלינו להשתמש בלולאת do-while.

```
public class DoWhile {
	public static void main(String[] args) {
		int cnt = 5;
		do {
			System.out.println(cnt);
			cnt--;
		}
		while (cnt > 0);
	}
}
```

בלולאת do-while מתבצעת איטרציה אחת של גוף הלולאה ללא בדיקת התנאי, ורק לאחר מכן נבדק התנאי שנמצא בתוך ה while. אם הוא מתקיים, תתבצע איטרציה נוספת, וכן הלאה.
עבור cnt=5 לא אמור להיות הבדל בין שתי הריצות. בדקו מה יקרה עבור cnt=0.

חלק ד' – לולאת for.

לולאת ה for גם היא מוכרת לכם מ Python, אבל המקבילה ה Java-אית ללולאת ה for מ Python היא לא לולאת ה for הקלאסית, אלא לולאת for each שאותה נראה בהמשך.
תחילה ללולאת ה for הקלאסית.
לולאה זו מורכבת מארבעה חלקים:

```
		for (int i = 0; i < 10; i++) {
			System.out.println(i);
		}
```

הגדרת משתנה עזר ואתחולו. המשתנה i מוגדר רק בתוך הלולאה, ואינו נגיש מחוץ לה. לו היינו רוצים ש i יהיה נגיש מחוץ ללולאה, יכולנו להגדיר אותו מחוץ ללולאה, ובתוך הלולאה רק לאתחל אותו ל 0 (או לכל ערך אחר).
לחילופין, ניתן היה לאתחל את i ל 0 מחוץ ללולאה ולהשאיר את החלק הצבוע בצהוב ריק.
חלק זה מתבצע פעם אחת בתחילת לולאת ה for.
בתוך חלק זה ניתן לאתחל יותר ממשתנה אחד. נראה דוגמא בהמשך.
בדיקת תנאי עצירה (נכון יותר – תנאי המשך). כל עוד התנאי מתקיים, ממשיכים. בתחילת לולאה, לאחר ביצוע בלוק האתחול נבדק תנאי העצירה. כמו בלולאת ה while, אם הוא לא מתקיים, יתכן מקרה שבו לולאת ה for לא תבצע אף איטרציה.

קידום משתנה העזר. שלב זה מבוצע לאחר ביצוע איטרציה אחת של גוף הלולאה. ניתן לקדם כמה משתנים בו זמנית (דוגמא בהמשך).
גוף הלולאה.

למעשה, לולאת ה for שקולה ללולאת while. את אותה הלולאה שרשמנו כאן. ניתן לבטא גם באמצעות המבנה של לולאת while:

```
int i = 0;
while (i < 10) {
	System.out.println(i);
	i++;
}
```

ההבדל בין שני המבנים האלה הוא שבלולאת ה for המשתנה i לא נגיש מחוץ לולאה, ובגירסת ה while הוא נגיש. יחד עם זאת, ניתן בקלות להתאים את לולאת ה for כך i יהיה נגיש גם מחוץ ללולאה, כפי שהוסבר קודם.
הריצו את התוכנית הבאה:

```
public class For {
	public static void main(String[] args) {
		int sum = 0;
		for (int i = 0, j = 5; i < 5; i++, j--) {
			sum += i * j;
		}
		System.out.println(sum);
	}
}
```

האם אתם מזהים את ארבעת אבני הבניין של הלולאה?
בתוך שורת ה for מופיעים כל מני ביטויים, ולכן כדי לחפש את שני המופעים של ; (נקודה פסיק). מה שמופיע לפני המופע הראשון של הסימן ; הראשון שייך להגדרת ואתחול משתנה העזר. בדוגמא זו מוגדרים שני משתני עזר.
בין המופע הראשון והשני של הסימן ; נמצא תנאי העצירה.
אחרי המופע השני של הסימן ; מופיע הקוד שמתקדם את משתני העזר. בדוגמא זו ניתן לראות כיצד מקודמים שני משתנים שונים.

כיצד נעבור על מערך בלולאת for?
אפשרות אחת:

```
double[] arr = { 1.0, 2.0, 3.0, 4.0, 5.0 };
double sum = 0.0;
for (int i = 0; i < arr.length; i++) {
	sum += arr[i];
}
System.out.println("arr sum is: " + sum);
```

הלולאה הזו בעצם עוברת על האינדקסים של המערך (מ 0 עד אורך המערך פחות 1).
אפשרות נוספת היא להשתמש בלולאת for each שהיא מזכירה מאוד את לולאת for שאתם מכירים מ Python. את לולאת ה for הקלאסית נחליף במבנה הבא:

```
for (double d : arr) {
	sum += d;
}
```

לולאת ה for each בעצם מהווה הפשטה של הלולאה הקלאסית: מאחורי הקלעים הקוד הזה מתורגם למבנה הקלאסי. זוהי דוגמא טובה להפשטה (abstraction) במובן שיש לנו תחביר שניתן יהיה בעתיד לממש אותו בצורה יעילה יותר מלולאת for קלאסית (למשל, מעבר מקבילי על המערך).

מה עושה הקוד הבא?

```
double[] arr = { 1.0, 2.0, 3.0, 4.0, 5.0 };
double sum = 0.0;
for (double d : arr) {
	d = 0;
}
```

התשובה היא: כלום. בכל איטרציה, המשתנה המקומי של הלולאה, d, מקבל השמה. אין לזה שום השפעה על המערך. לו היינו רוצים לעדכן את המערך, היינו צריכים לגשת לאינדקס מסויים ולעדכן דרכו. את זה ניתן לעשות באמצעות לולאת ה for הקלאסית שלמעשה עוברת על האינדקסים.

חלק ה – break ו continue

את משפט ה break ראינו כבר ב switch/case וכעת נראה את השימוש הנפוץ יותר שלו.
משפט break – מפסיק את הלולאה שבתוכה נמצא המשפט
משפט continue – מפסיק את האיטרציה הנוכחית וממשיך לאיטרציה הבאה.
הריצו את הקוד הבא:

```
public class BreakContinue {
	public static void main(String[] args) {
		for (int i = 0; i < 5; i++) {
			if (i == 3) {
				break;
			}
			System.out.println("*: " + i);
		}

		for (int i = 0; i < 5; i++) {
			if (i == 3) {
				continue;
			}
			System.out.println("**: " + i);
		}
	}
}
```

מה יודפס?
בלולאה הראשונה, ה break מפסיק את הלולאה ברגע מגיעים לערך i=3.
בלולאה השניה, כאשר i=3 מתבצע משפט ה continue ולכן האיטרציה הנוכחית מסתיימת והלולאה ממשיכה לאיטרציה נוספת.

ומה קורה במקרה של לולאה מקוננת?
הריצו את הקוד הבא:

```
public class BreakContinue2 {
	public static void main(String[] args) {
		for (int i = 1; i < 5; i++) {
			for (int j = 1; j < 5; j++) {
				/* if (i*j >= 9) {
					break;
				}*/
				System.out.print(i*j + "\t");
			}
			System.out.println();
		}
	}
}
```

הקוד הבא מדפיס את הלוח הכפל של המספרים מ 1 עד 4. מה יקרה אם נוציא את בלוק ה if מהלולאה? אנחנו רואים שהפקודה break מפסיקה את הלולאה הפנימית והלולאה החיצונית ממשיכה לרוץ.
האם ניתן להפסיק באמצעות פקודת break את שתי הלולאות? כלומר, לעצור את הריצה של שתיהן ברגע שהגענו למכפלה שהיא גדולה או שווה ל 9? אפשר כמובן להגדיר משתנה עזר בוליאני שיאותחל בלולאה הפנימית ויבדק בחיצונית, אבל יש פתרון תחבירי פשוט יותר:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/branch.html