# שרשרת חוליות גנרית

**יצירת שרשרת והצגתה**

לעיתים קרובות מתחיל התרגיל במשפט:
"נתונה שרשרת חוליות ... כתוב פעולה שתקבל את השרשרת ותבצע ..."

מטרת התרגיל: לכתוב את הפעולה הנדרשת.
הבעיה - כדי לבדוק את נכונות הפתרון שכתבנו, יש לבנות את השרשרת.
פעילות מייגעת החוזרת על עצמה עבור כל תרגיל שנרצה לפתור ולהריץ במחשב.
ואם התכנית מכילה שרשרת חוליות מטיפוסים שונים, נצטרך לכתוב את הקוד עבור כל שרשרת בנפרד.

כדי לחסוך בשכפול קוד, נוכל להמיר את הפעולות הקונקרטיות בפעולות גנריות.

**שים ♥:** אין מחלקה המגדירה שרשרת חוליות. השרשרת הינה מבנה נתונים, ובמבנה נתונים מטפלים באמצעות פעולות סטטיות.

כדי להפוך פעולה קונקרטית (Node<Integer>) לפעולה גנרית: נוסיף למילה static את הסימון <T>:

**public static <T>** טיפוס-ערך-מוחזר methodName (רשימת הפרמטרים שהפעולה מקבלת)

אם הפרמטר מתייחס לעצמים, נרשום את טיפוס העצם כ- T:

- עצם יחיד T
- מערך מסוג העצם T [ ]
- שרשרת או הפנייה Node<T>

למשל: הפעולה show מקבלת שרשרת חוליות גנרית chain
הפעולה getChain מקבלת מערך גנרי arr

**שים ♥:** כל האמור בדף זה נכון לתרגול ליד המחשב.
בבחינת הבגרות נכתוב את הפעולה הקונקרטית (המשתמשת בטיפוס הנתון בשאלה)

התכנית שלהלן יוצרת שתי שרשראות של חוליות, האחת שרשרת של מספרים שלמים והאחרת של מחרוזות.
הבנייה וההצגה נעשות באמצעות **פעולות סטטיות גנריות**:

```
public class ChkNodeT
{
	/*		שימוש בפעולות סטטיות גנריות		*/
	public static void main(String[] args)
	{
		//--- המערך המכיל את הערכים כפי שניתנו בתרגיל --
		Integer [] arr1 = {3, 3, 4, 5, 3, 7, 3, 3, 5, 8, 3};
		String [] arr2 = {"aaa", "bbb", "ccc", "ddd", "eee"};

		Node <Integer> p1 = getChain(arr1);
		Node <String> p2 = getChain(arr2);

		show (p1);
		show (p2);
	}

	//---  פעולה המציגה את השרשרת ---
	public static<T> void show (Node<T> chain)
	{
		System.out.print("[");
		while (chain != null)
		{
			System.out.print(chain.getValue().toString());
			if (chain.hasNext())
				System.out.print(", ");
			else
				System.out.println("]");
			chain = chain.getNext();
		}
	}

	//--- הפעולה מקבלת מערך מטיפוס גנרי, יוצרת ומחזירה שרשרת חוליות המכילות ערכים אלו ---
	public static<T> Node<T> getChain (T [] arr)
	{
		int n = arr.length;

		//--- בניית השרשרת ---
		Node<T> chain = new Node<T> (arr[0]);
		Node<T> pos = chain;

		for (int i = 1 ; i < n ; i++)
		{
			pos.setNext(new Node<T> (arr[i]));
			pos = pos.getNext();
		}
		return chain;
	}

}
```

שים לב לשימוש במחלקה העוטפת Integer לבניית מערך של מספרים שלמים.
את הטיפוס הגנרי T ניתן להחליף אך ורק בעצם (מחלקה). לא בטיפוס בסיסי

הסימון הצהוב נועד לצרכי הדגשה בלבד ואינו מופיע בקוד

# מחלקת שרות גנרית

נוכל ליצור מחלקת שרות שתכיל את הפעולות הגנריות, ולקרוא לפעולות שלה מתוך התכנית:

```
/*        מחלקת שרות לשרשרת חוליות גנרית        */
public class Chain
{
        //---  פעולה המציגה את השרשרת ---
        public static<T> void show (Node<T> chain)
        {
                System.out.print("[");
                while (chain != null)
                {
                        System.out.print(chain.getValue().toString());
                        if (chain.hasNext())
                                System.out.print(", ");
                        else
                                System.out.println("]");
                        chain = chain.getNext();
                }
        }
        //--- הפעולה מקבלת מערך מטיפוס גנרי, יוצרת ומחזירה שרשרת חוליות המכילות ערכים אלו ---
        public static<T> Node<T> getChain (T [] arr)
        {
                int n = arr.length;

                //--- בניית השרשרת ---
                Node<T> chain = new Node<T> (arr[0]);
                Node<T> pos = chain;
                for (int i = 1 ; i < n ; i++)
                {
                        pos.setNext(new Node<T> (arr[i]));
                        pos = pos.getNext();
                }
                return chain;
        }
}
```

**התכנית:**

```
public class ChkGenericChain
{
        public static void main(String[] args)
        {
                Integer [] arr1 = {3, 3, 4, 5, 3, 7, 3, 3, 5, 8, 3};
                String [] arr2 = {"aaa", "bbb", "ccc", "ddd", "eee"};

                Node <Integer> p1 = Chain.getChain(arr1);
                Node <String> p2 = Chain.getChain(arr2);

                Chain.show (p1);
                Chain.show (p2);
        }
}
```

את הפעולות מזמנים ע"י ציון שם המחלקה המכילה אותן.

הסימון הצהוב נועד לצרכי הדגשה בלבד ואינו מופיע בקוד

## פעולות גנריות התלויות בסוג האיבר

יש פעולות העוסקות בתוכן החוליה: הימצאות x בשרשרת, מיון הערכים בשרשרת וכד'.

כדי להשוות את תוכן החוליה לערך, נשתמש בפעולה **equals** המחזירה אמת אם הערכים שווים ושקר אחרת.

כדי למיין את החוליות לפי סדר מסוים, נשתמש בפעולה **compareTo** המחזירה מספר חיובי אם הערך המשווה (המפעיל) גדול מהערך המושווה (הפרמטר), מספר שלילי אם הוא קטן ממנו ו-0 (אפס) אם הם שווים:

```
//--- פעולה המחזירה אמת אם הערך x נמצא בשרשרת, ושקר אחרת ---
public static<T> boolean exist (Node<T> chain, T x)
{
	while (chain != null)
	{
		if(chain.getValue().equals(x))
			return true;
		chain = chain.getNext();
	}
	return false;
}
```

**זימון הפעולה בתכנית:**

```
System.out.println("exist(p1, 8) : " + Chain.exist(p1, 8));
System.out.println("exist(p2, 'ggg') : " + Chain.exist(p2, "ggg"));
```

**הפלט:**

```
exist(p1, 8) : true
exist(p2, 'ggg') : false
```

שתי הפעולות מוגדרות בכל **המחלקות העוטפות** של הטיפוסים הבסיסיים.

| | | |
|---|---|---|
| **int** x, y; | x.equals(y) … | שגוי |
| **Integer** x, y; | x.equals(y) … | תקין |

הפעולה equals מוגדרת באופן אוטומטי (הורשה) בכל מחלקה שנכתוב, אבל משווה הפניות ולא את מצב העצם (הפעולה תחזיר אמת אם שתי ההפניות מפנות לאותו עצם, ושקר אחרת, גם אם ערך התכונות זהה).

הפעולה compareTo אינה מוגדרת במחלקה שכתבנו, ויהיה עלינו לכתוב אותה במחלקה.

כדי להשתמש בפעולות הגנריות בהצלחה, נוסיף את שתי הפעולות למחלקות המגדירות את העצם.