



נושאים מתקדמים בתכנות מונחה עצמים ושפת JAVA

ד"ר רינה צביאל-גירשין

יחידה 1

סקירה כללית של הקורס



■ מעבר משפת C ו C# לשפת JAVA

■ C לא OOP

■ C# היא pure OOL וגם מאוד דומה לJAVA

■ נושאים מתקדמים בתכנות מונחה עצמים

היסטוריה של JAVA

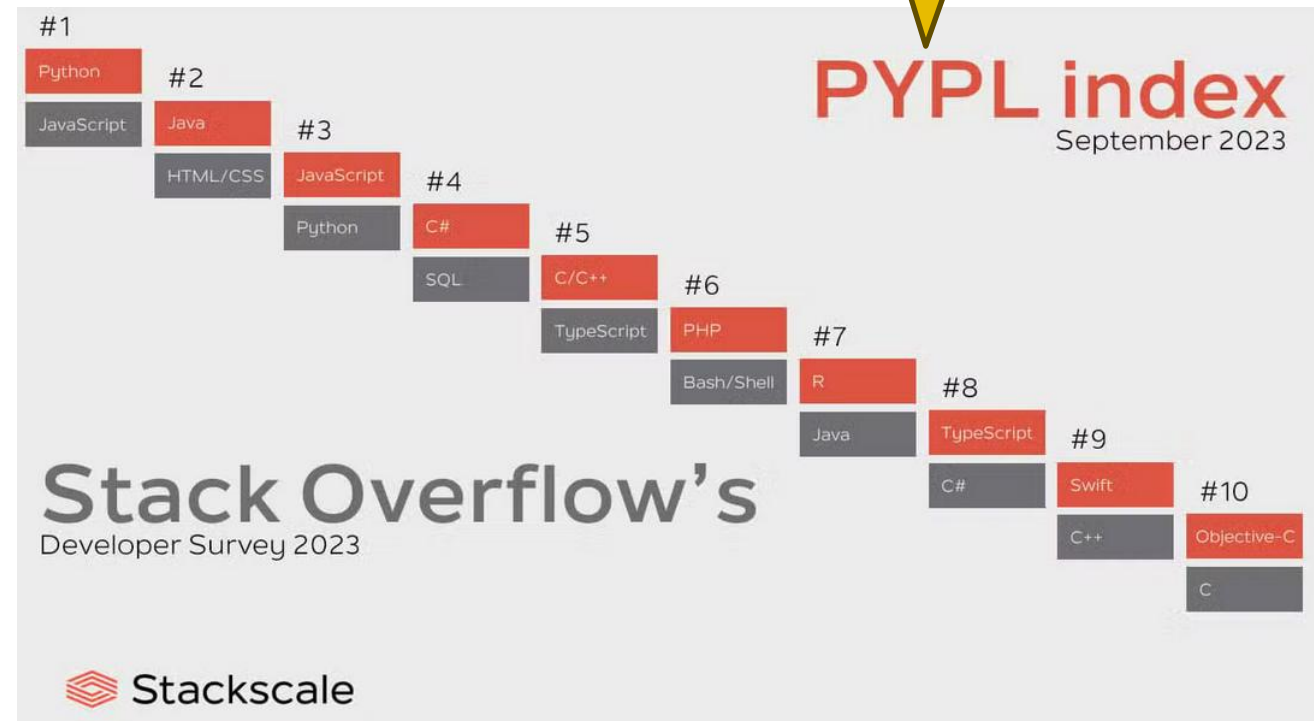
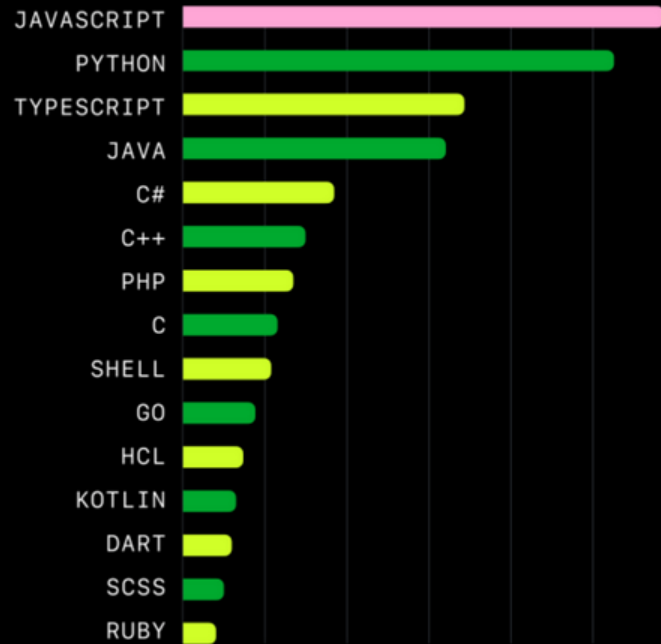


- פותחה במקור על ידי צוות קטן במעבדות Sun Microsystems בהובלת James Gosling
- 1995 שינוי שם מ Oak ל Java
- 1996 גרסה ראשונה Java1.0
- עקרון Write Once, Run Anywhere
- 2009 נרכשה על ידי Oracle

מה קורה בתחום

How often language tutorials are searched on Google.

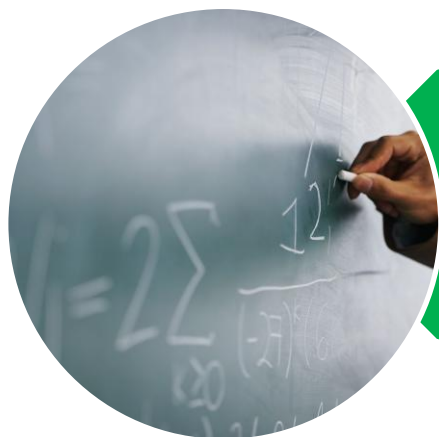
Top 10 Programming Languages on GitHub in 2023



פרטים טכניים



אתר הקורס – מודל



הרצאה 2 ש"ש

תרגול 2 ש"ש

פרטי מידע

■ חומר נמצא באתר הקורס

■ <http://ruppinet.ruppin.ac.il>

■ <https://moodle.ruppin.ac.il>

■ ספר

- H.M. Deitel & P.J. Deitel, *Java: How to Program*, 11th ed. (Prentice Hall, 2018) or any previous editions

■ כל ספר אחר בעברית או אנגלית

■ אתרים

■ אינסוף...רק לרשום "Java tutorial"

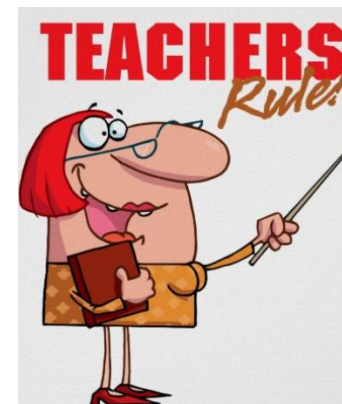


הוראה וצוות הקורס

מרצה

ד"ר רינה צביאל-גירשין

rinazg@ruppin.ac.il



מתרגלים

גב מעין זנו

maayan24nasimi@gmail.com

שי אברהם

shaykos@outlook.com



Grading policy



תרגילי בית

10%

**חלקם חובה
עבודה בזוגות**



2 בחנים

בכיתה

**חובה נוכחות פיזית
20%**



מבחן

70%

מדיניות הציונים



- בחינת סוף הסמסטר - 70%

- חובת מעבר במבחן

- תרגילי בית

- כל שבוע, שבועיים (5-6)

- N-1 הכי טובים

- הגשה בזוגות

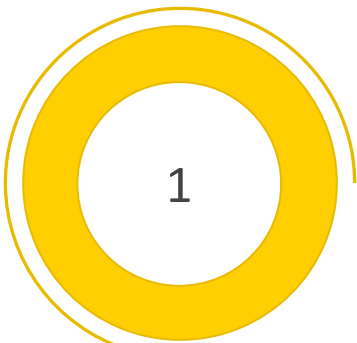
- הגשה דרך האתר MOODLE וגם חובה להשתמש בGITHUB

מפת הקורס

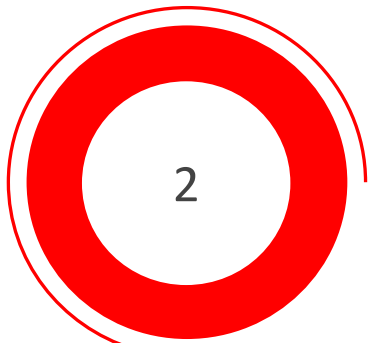


שבוע 3 יחידה היברידית

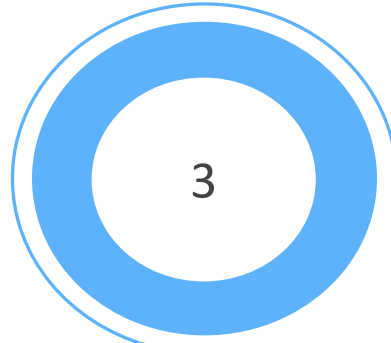
שבוע 9 יחידה היברידית



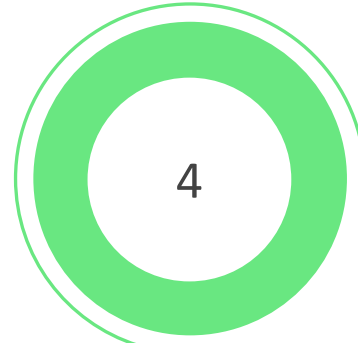
מעבר בין
C# ל JAVA



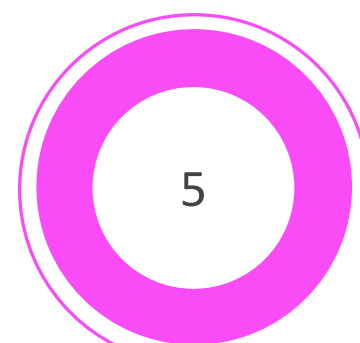
רשתות
תקשורת שרת לקוח



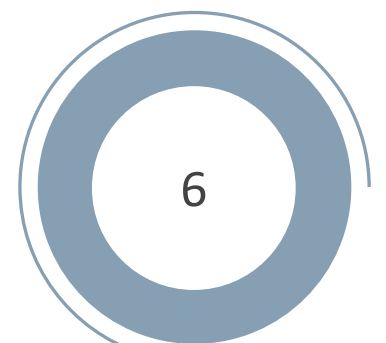
תהליכון
ריבוי תהליכונים
ניהול תהליכים



GUI
רכיבים
אירועים



design
patterns
תבניות



תכנות גנרי
ואספים
עבודה עם
בסיסי נתונים
JDBC

מבט על על היחידה



- Java-ו C#
- מחלקה
- סטאטי
- אובייקט מוכל
- חריגות
- הורשה
- class Object
- מחלקה אבסטראקטית
- ממשק
- חריגות
- javadoc

JAVA מול C#



- קל ללמוד כל אחת מן השפות למי שיש רקע בשפה השנייה
- שתי שפות מאוד דומות
- בתחביר syntax (יש יותר דמיון משוני)
- מרבית מבני בקרה דומים
- שתי שפות OOP שתומכות בהורשה inheritance, כימוס encapsulation, רב-צורתיות polymorphism
- ניתן להשתמש/להרחיב הממשקים interface
- ניהול זכרון על ידי garbage collection

C# מול JAVA

2000

1995

תכונה	C# (.NET)	JAVA
OS	Windows .NET core cross-platform	Multiple
Runtime	CLR	JVM
Web Server Scripting	ASP.NET WebForms	JSF
Data Access	ADO.NET	JDBC
ORM (object-relational mapping)	Entity Framework	Hibernate (add-on)
Applications	Windows app development, game development(unity), web development	Android app development, enterprise applications, web development



השוני



- ריבוי מערכות platform independent

- *write once, run anywhere*

- Libraries and Frameworks

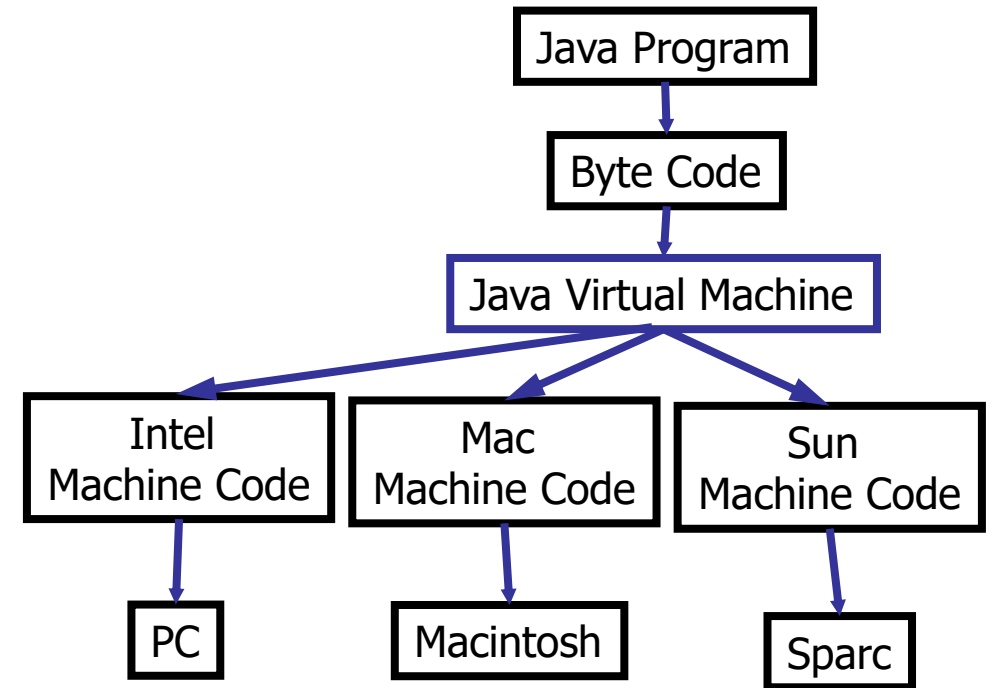
- C# - .NET framework

- Java - set of extensive libraries and frameworks, such as Spring, Hibernate, and JavaFX

- אין תכונות (getter setter) properties

מספר מילים על Byte Code ו JVM

- קוד מתורגם לבית קוד Byte Code
- למעשה זו שפת מכונה עבור JVM מכונה מדומה
- JVM מפרש interpret בית קוד לשפת מעבד עבור כל מערכת הפעלה OS



Java Main Environments



■ צורת כתיבת קוד :

Operating System (Applications) ■

Web browser (Applets) ■

סביבת עבודה



■ קיימות מספר סביבות עבודה

■ נשתמש ב-Eclipse

■ תהליך דו-שלבי

■ התקנת JAVA (JDK)

■ יכול להיות מותקן

■ $JDK = JRE + Developer\ tools$

■ $JRE = JVM + library\ classes$

■ התקנת Eclipse

טיפוסים



C# (type/alias)	Java (type/wrapper)
bool / System.Boolean	boolean / java.lang.Boolean
byte / System.Byte	byte / java.lang.Byte
short / System.Int16	short / java.lang.Short
int / System.Int32	int / java.lang.Integer
long / System.Int64	long / java.lang.Long
float / System.Single	float / java.lang.Float
double / System.Double	double / java.lang.Double
char / System.Char	character / java.lang.Character

טיפוסים פרימיטיביים ■ primitive data types

byte, short, int, long, float, double, boolean and char ■

לא פרימיטיביים ■

non-primitive data type or object data type

Classes - String, Array ■

פקודות קלט פלט

עבור פקודות קלט/פלט יש להשתמש ב `System`

פלט

`System.out.print()` or `System.out.println (parameters)`

קלט

`Scanner` נמצא בחבילה `java.util`

```
Scanner scanner = new Scanner(System.in);
```

```
String name = scanner.nextLine();
```

```
int age = scanner.nextInt();
```

רצוי לסגור בסוף

לצרף ספריה/חבילה `import`

התוכנית הבסיסית

JAVA

```
import java.io.*;

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

Hello JAVA!

Filename: HelloWorld.java

C#

```
using System;

class HelloWorld
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello C#!");
    }
}
```

קבועים



```
private readonly int GRADE=100;
```

```
private final int GRADE=100;
```

■ readonly #C

■ Java final

מבני בקרה



- תנאי if/else
- רב בחירה switch(){case X:...break;... default}
- לולאות while, do .. while, for
- אין foreach
- אבל for (Type element : collection) {///...}

```
import java.util.Scanner;
```

```
public class Example
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Enter the number of elements: ");
```

```
        int size = scanner.nextInt();
```

```
        int[] numbers = new int[size]; //array
```

```
        for (int i = 0; i < size; i++)
```

```
        {
```

```
            System.out.print("Enter element " + (i + 1) + ": ");
```

```
            numbers[i] = scanner.nextInt();
```

```
        }
```

```
        System.out.println("You entered the following numbers:");
```

```
        for (int number : numbers) //instead of for each
```

```
            System.out.println("Number: " + number);
```

```
        scanner.close();
```

```
    }
```

```
}
```

דוגמה

```
terminated / app / java Application / C:\Users\pa  
Enter the number of elements: 4  
Enter element 1: 12  
Enter element 2: 45  
Enter element 3: 56  
Enter element 4: -2  
You entered the following numbers:  
Number: 12  
Number: 45  
Number: 56  
Number: -2
```

מחרוזות



- **מחרוזת String** היא אובייקט שמייצג רצף של תווים.
- טיפוס מסוג הפניה.
- **דוגמה:**

```
String name = "Rina";
```

- בדומה לC# מחרוזת היא – **immutable** כלומר, **בלתי ניתנת לשינוי** לאחר שהיא נוצרה.

■ **דוגמה:**

```
String name = "Rina";
```

```
String upper = name.toUpperCase(); //RINA
```


דוגמה

■ דוגמה:

```
String name = "Rina";  
name = name + " Zviel";
```

- בדוגמה זו **לא שונתה** המחרוזת המקורית Rina
- **נוצרה** מחרוזת חדשה Rina Zviel ומשתנה name מצביע עליה
- המחרוזת המקורית Rina נשארה בזיכרון עד שאוסף זבל אוטומטי Garbage Collector ייאסף אותה.

String מתודות נפוצות



- `substring(int beginIndex, int endIndex)` – חלק מהמחרוזת

```
String word = "Programming";  
System.out.println(word.substring(0, 6)); // Progra
```

- `equals()` – השוואת מחרוזות לפי תוכן

```
String a = "Java";  
String b = "java";  
System.out.println(a.equals(b)); // false  
System.out.println(a.equalsIgnoreCase(b)); // true
```

- `charAt(int index)` – קבלת תו לפי מיקום

```
String s = "Code";  
System.out.println(s.charAt(1)); // o
```

- `length()` – אורך המחרוזת (מספר תווים)

```
String name = "Rina";  
System.out.println(name.length()); // 4
```

- `trim()` – הסרת רווחים מיותרים בתחילת ובסוף המחרוזת

```
String text = " hello world ";  
String trimmed = text.trim();  
System.out.println(trimmed); // "hello world"
```

- `toUpperCase()` – המרת כל האותיות לאותיות גדולות

```
String msg = "good hour";  
System.out.println(msg.toUpperCase()); // GOOD HOUR
```

- `toLowerCase()` – המרת כל האותיות לאותיות קטנות

```
String msg = "HELLO";  
System.out.println(msg.toLowerCase()); // hello
```

מחלקה StringBuilder




- מחרוזת היא בלתי ניתנת לשינוי – immutable כאשר StringBuilder היא מחלקה mutable ניתנת לשינוי.
- באמצעות מחלקה StringBuilder ניתן לשנות את התוכן של המחרוזת בלי ליצור אובייקט חדש בכל פעם.
- **דוגמה:**

```
StringBuilder sb = new StringBuilder();  
sb.append("Hello");      // מוסיף טקסט לסוף  
sb.append(" ");  
sb.append("World!"); // Hello World!
```

יעילות

```
public class StringTest
{
    public static void main(String[] args)
    {
        long start = System.currentTimeMillis();
        String result = "";
        for (int i = 0; i < 10000; i++)
            result += i; // new string

        long end = System.currentTimeMillis();
        System.out.println("Time using String: " + (end -
start) + " ms");
    }
}
```



```
public class StringTest
{
    public static void main(String[] args)
    {
        long start = System.currentTimeMillis();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 10000; i++)
            sb.append(i); // the same string

        long end = System.currentTimeMillis();
        System.out.println("Time using String: " + (end -
start) + " ms");
    }
}
```

StringBuilder יותר מהירה כי היא לא יוצרת אובייקט חדש בכל איטרציה.

יעילות



- כאשר יש קוד שמחבר הרבה מחרוזות (כמו בתוך לולאה), השימוש ב־String יוצר אובייקט חדש בכל פעם, מה שפוגע בביצועים.
- לעומת זאת, `StringBuilder` משנה את התוכן באותו אובייקט, ולכן היא יעילה יותר בזיכרון ובמהירות

מתודות נפוצות StringBuilder



תיאור	מתודה
מוסיף טקסט לסוף	<code>append(String s)</code>
מוסיף טקסט במיקום מסוים	<code>insert(int offset, String s)</code>
מוחק תווים בין מיקומים	<code>delete(int start, int end)</code>
הופך את סדר התווים	<code>reverse()</code>
מחליף תווים בתחום מסוים	<code>replace(int start, int end, String s)</code>

StringBuilder יותר מהירה כי היא לא יוצרת אובייקט חדש בכל איטרציה.

מערכים

■ אין הבדל בהכרזה, גישה

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 }; //or  
int[] numbers = { 1, 2, 3, 4, 5 }; //or  
int[] numbers;  
numbers = new int[5];  
...  
numbers[2] = 10;
```

■ #C Length ב-JAVA length

```
String[] names = {"Rina", "Avner", "Beni", "Arava"};  
for (int i = 0; i < names.length; i++)  
    System.out.println(names[i]);
```

דוגמה

```
public class mainClass
{
    public static void main(String[] args)
    {
        String[] names = {"Rina", "Avner", "Beni", "Arava"};
        for (int i = 0; i < names.length; i++)
            System.out.println(names[i]);
    } //end of main
} //end of mainClass
```

```
terminated
Rina
Avner
Beni
Arava
```


מערך רב ממדי



■ תחביר

```
dataType[1st dim][2nd dim][...]..[Nth dim] aName = new datatype[size1][size2]....[sizeN];
```

■ דוגמה:

```
int[][][] arr3D = new int[10][20][30];
```

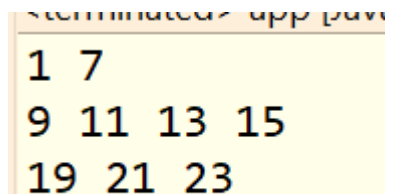
■ ניתן ליצור מערך רב ממדי בגודל לא סימטרי

```
int[][] oddNumbers = { {1, 7}, {9, 11, 13, 15}, { 19, 21, 23} };
```

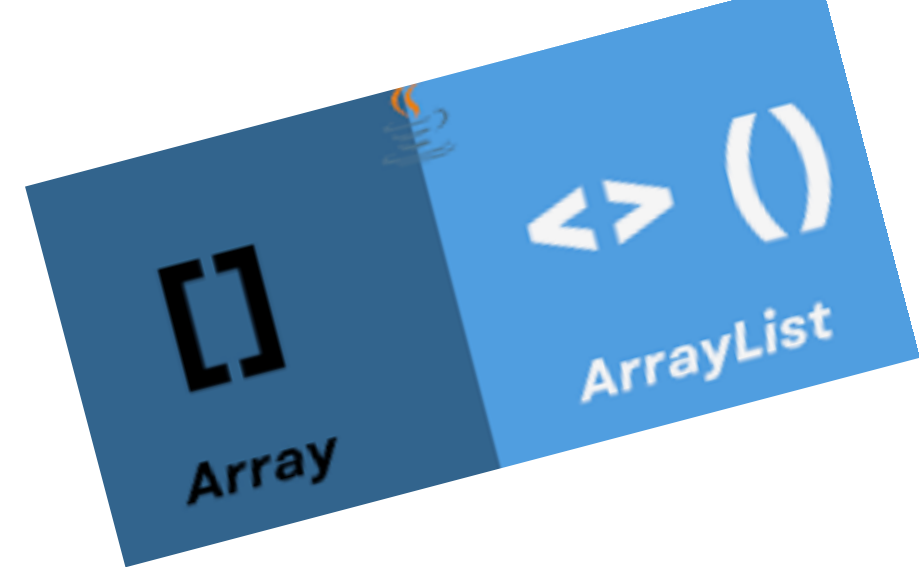
דוגמה

```
public class mainClass
{
    int[][] numbers = { {1, 7},
                        {9, 11, 13, 15}, { 19, 21, 23} }; //different size

    for(int i = 0; i < numbers.length; i++)
    {
        for(int j = 0; j < numbers[i].length; j++)
            System.out.print(numbers[i][j] + " ");
        System.out.println( );
    }
} //end of main
} //end of mainClass
```



```
1 7
9 11 13 15
19 21 23
```



מערך דינאמי ArrayList

- מערך דינאמי בעל גודל משתנה ArrayList
- נמצא בחבילה java.util
- יש ליצור (חשוב לציין טיפוס האיברים)

```
ArrayList<TYPE> names = new ArrayList<TYPE>();
```

- להוסיף איברים

```
names.add(NewItem);
```

- ניתן לגשת לאיבר במקום i get/set

```
names.get(0);    names.set(0, "Beni");
```

- יש לו גודל

```
names.size();
```

דוגמה

```
public class mainClass
{
    public static void main(String[] args)
    {
        ArrayList<String> names = new ArrayList<String>();
        names.add("Anver");
        names.add("Rina");
        names.add("Shiri");
        System.out.println(names);
        System.out.println("The size is " + names.size());
        System.out.println("Contains Beni? " + names.contains("Beni"));
        names.clear(); //clears
    } //end of main
} //end of mainClass
```

```
terminated: app [java / application]
[Anver, Rina, Shiri]
The size is 3
Contains Beni? false
```

מחלקה פשוטה



- מחלקה שיוצרת אובייקט

```
public class NAME
{
    //data fields;
    //methods;
}
```

- הרשאות גישה - ברירת מחדל לא פרטי "חבילתי" "package only" כל המחלקות באותה חבילה רואות את השדה



חבילה/מארז

- JAVA מגיעה עם אוסף גדול של מחלקות סטנדרטיות לשימוש המתכנת.
 - המחלקות מאורגנות במבנה של מארז (package).
 - החבילה/המארז `java.lang` מכילה אוסף מחלקות בסיסיות של השפה.
 - למשל את המחלקות `System`, `Math`, `String`
 - מארז זה **מוכל אוטומטית** בכל תוכנית בג'אווה.
-
- אם רוצים לכלול גם מחלקות ממארזים אחרים, יש להשתמש בפקודה `import`
 - דוגמה:

```
import java.util.ArrayList;
```

Package vs namespace

JAVA

מארגן קבצים פיזיים

```
package SomeExample;
```

```
public class Example  
{  
}
```

similar

C#

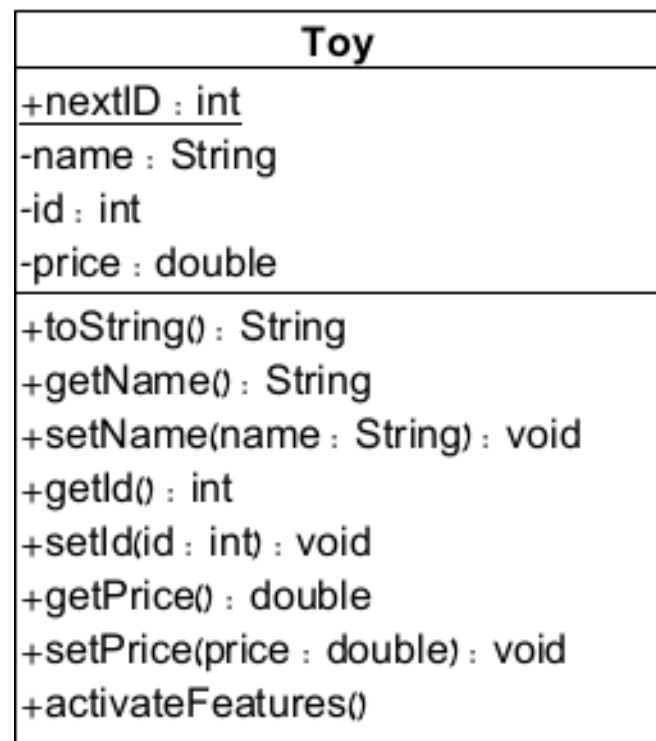
מארגן יחידות לוגיות
כגון מחלקות וממשקים

```
namespace SomeExample;
```

```
public class Example  
{  
}
```

- לא חובה להגדיר מארז
- אבל אם מגדירים אז זו צריכה להיות שורה ראשונה בקובץ
- אין קינון nesting של החבילות

מחלקה



מחלקה פשוטה

Toy

```
public class Toy
{
    // Static variable to keep track of the next ID to be assigned
    private static int nextId = 1;

    private String name;
    private int id;
    private double price;

    public Toy(String name, double price)
    {
        this.name = name;
        setPrice(price); // Use setter to ensure validation
        // Assign the next available ID and increment it
        this.id = nextId++; //no setID
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getId()
    {
        return id;
    }
}
```

```
public double getPrice()
{
    return price;
}

public void setPrice(double price) throws IllegalArgumentException
{
    // If negative throw Exception
    if (price < 0)
        throw new
            IllegalArgumentException("Price cannot be negative.");
    this.price = price;
}

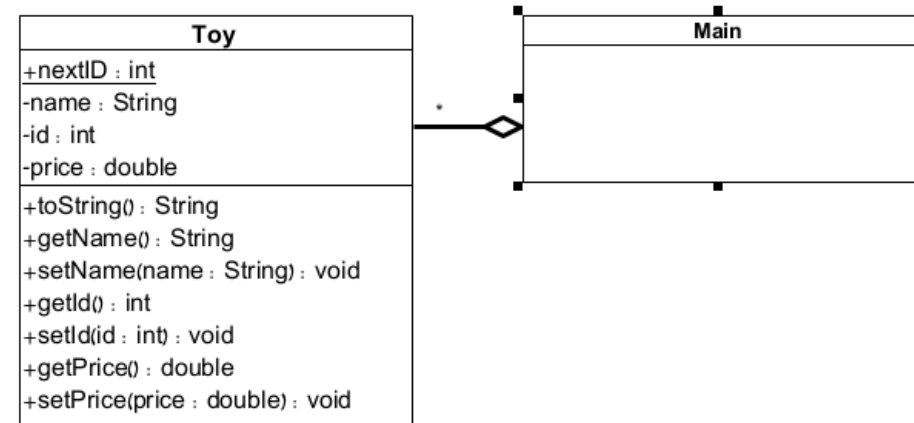
public void activateFeatures()
{
    System.out.println("Activating basic features of the toy: " + name);
}

@Override public String toString()
{
    return "Toy: " + name + " - ID: " + id + " - Price: $" + price;
} //end of class
```

```

public class mainClass
{
    public static void main(String[] args)
    {
        try {
            // Create some Toy objects
            Toy toy1 = new Toy("Dog", 25.0);
            System.out.println(toy1);
            Toy toy2 = new Toy("Dog", -45.2);
            System.out.println(toy2);
        }
        catch (IllegalArgumentException e)
        {
            System.out.println("Exception caught: " + e.getMessage());
        }
    } //end of main
} //end of mainClass

```



```

Toy: Dog - ID: 1 - Price: $25.0
Exception caught: Price cannot be negative.

```

main

```
public class MainClass
{
    public static void main(String[] args)
    {
        // Create some Toy objects
        Toy toy1 = new Toy("Dog", 25.0);
        System.out.println(toy1);
        Toy toy2 = new Toy("Dog", -45.2);
        System.out.println(toy2);
    } //end of main
} //end of mainClass
```

כתיבה לא מסודרת
ללא try catch

```
<terminated> app [Java Application] C:\Users\para\Downloads\eclipse-java-2020-12-R-win32-x86_64\ec
Exception in thread "main" Toy: Dog - ID: 1 - Price: $25.0
java.lang.IllegalArgumentException: Price cannot be negative.
    at Toy.setPrice(Toy.java:82)
    at Toy.<init>(Toy.java:34)
    at app.main(app.java:14)
```

גישה לשדה בmain

```
public class Toy
{
    ...
    private String name;
    double price; //not private
}
```

חלק מmain

```
// Create some Toy objects
Toy toy1 = new Toy("Dog", 25.0);
System.out.println(toy1.price);
System.out.println(toy1.name);
```

Description

✖ Errors (1 item)

💡 ✖ The field Toy.name is not visible



מספר מילים על מחלקות



- מחלקות של רמה עליונה top-level הם
 - public או כלום (non public)
- לא ניתן ליצור מחלקה (לא פנימית inner) שהיא פרטית private
- אם לא רושמים כלום אז חברי מחלקה members ניתנים לגישה רק לחבילה/מארז
- **תזכורת:** ברירת מחדל של הרשאות גישה - לא פרטי "חבילתי"

One per class

מספר מילים על static

- משתנה סטאטי – אחד משותף לכל האובייקטים של המחלקה

```
private static int nextId = 1;  
this.id = nextId++;
```

- מתודה סטאטית – מתודה ששייכת לכל המחלקה ולא לאובייקט ספציפי

```
public static int MethodName(int number)  
{...}
```

- שימוש/קריאה

```
className.MethodName();
```

- מותר להשתמש בדרך אובייקט (אבל לא מומלץ warning)

- **בצ'אסור** להפעיל מתודה סטאטית דרך אובייקט.

```
public class Utility
{
    // Static variable
    public static final double PI = 3.14159;

    // Static method
    public static int square(int number)
    { return number * number; }

    public static void main(String[] args)
    {
        // Accessing static variable
        System.out.println("Value of PI: " + Utility.PI);

        // Calling static method
        int result = Utility.square(5);
        System.out.println("Square of 5: " + result);
    }
} //end of class
```

Value of PI: 3.14159
Square of 5: 25

מחלקה סטאטית



- מחלקה סטאטית חייבת להיות פנימית מקוננת
- ניתן לגשת רק לחברים סטאטיים של מחלקה חיצונית

```
class OuterClass
{
    static class NestedDemo
    {
    }
}
```



```
public class StaticExample
{
    private static String s= "Static String";
```

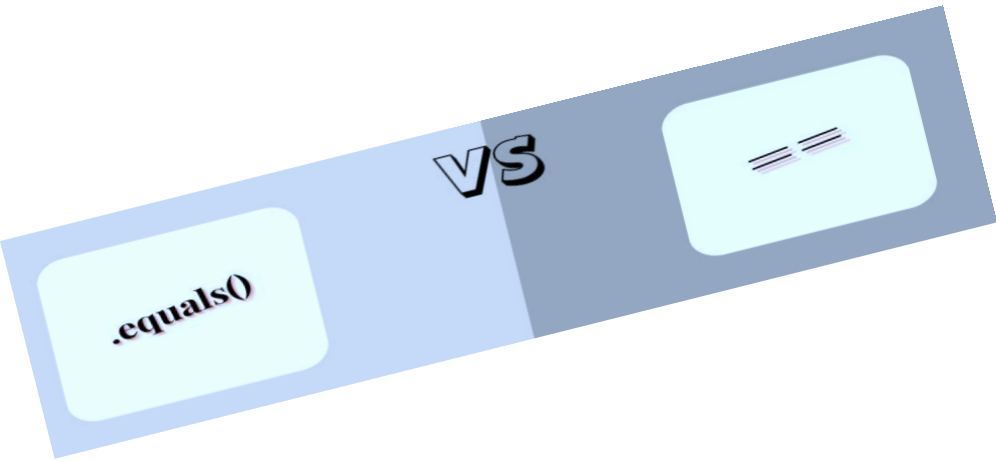
```
static class NestedDemo
```

```
{
    //non-static method of the nested class
    public void show()
    { System.out.println(s); }
} //end of NestedDemo class
```

```
public static void main(String args[])
```

```
{
    StaticExample.NestedDemo obj = new StaticExample.NestedDemo();
    //invoking the method of the nested class
    obj.show();
}
} //end of class
```

```
<terminated> StaticExam
Static String
```



== vs equals

- בC# == משווה וניתן להשתמש בו במקום **E**quals
- בJAVA == מיצר תוצאות שונות ולכן להשוואה עדיף להשתמש במתודה equals (חוץ ממחלקת String)
- אופרטור == משווה כתובות בזכרון
- מתודה **e**quals משווה תוכן

Exceptions



- חריגה – שגיאת הרצה (הקוד התקמפל אבל ...)
- זורקים חריגה `throw`
- תופסים חריגה `catch`
- בלוק מסכם `finally`
- קורה תמיד

- בחתימה של מתודה שזורקת חריגה ולא תופסת צריך לכתוב `throws`
`public type method() throws SomeException {}`
`public void setPrice(double price) throws IllegalArgumentException {}`

```

import java.util.Scanner;

public class ExcExample
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter which element to print: ");
        int num = scanner.nextInt();
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[num]);
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        } finally
        {
            System.out.println("The 'try catch' is finished.");
        }
        scanner.close();
    }
} //end of class

```

חריגה

```

Enter which element to print: 2
3
The 'try catch' is finished.

```

```

Enter which element to print: 6
Something went wrong.
The 'try catch' is finished.

```

C#

```
public class A : B
{
    ...
    base();
    ...
}
```

```
public class A extends B
{
    ...
    super();
    ...
}
```

הורשה

- single inheritance - אב יחיד
- extends יש להרחיב מחלקת בסיס
- super קריאה למחלקת בסיס באמצעות
- תחביר:



דוגמה להורשה בסיסית

JAVA

```
public class A
{
    public A ( string name)
    {
    }
}

public class B extends A
{
    public B ()
    { this ("Rina – Java" , 0 ); }

    public B (string name, int type)
    { super(name); }
}
```

C#

```
public class A
{
    public A ( string name)
    {
    }
}

public class B:A
{
    public B (): this ("Rina – C#" , 0 )
    {
    }
    public B (string name, int type): base(name)
    {
    }
}
```

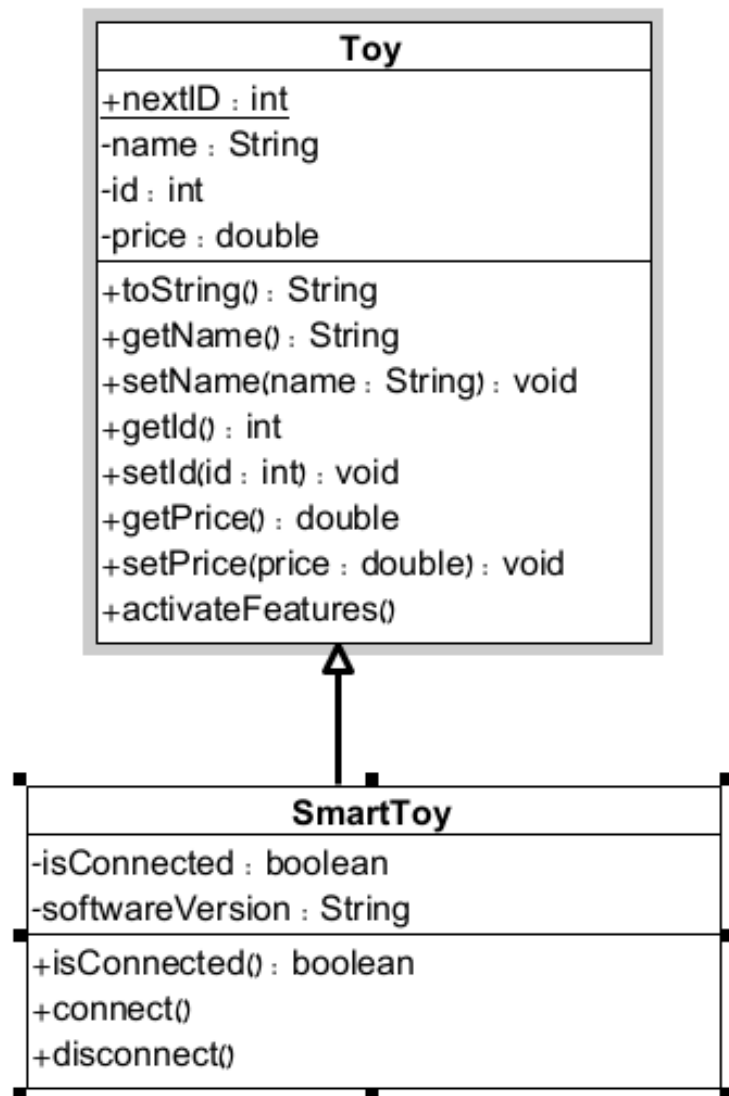
מתודה מדומה



- כל המתודות ב-JAVA מדומות תמיד (ברירת מחדל default).
- לכן תמיד ניתן לדרוס אותן `override`
- במקרה של דריסה מקובל לרשום

@Override

נחזור למחלקת TOY



חנות הצעצועים "Toys is us" מכרה עד כה בעיקר צעצועים קלאסיים עם מאפיינים בסיסיים – כגון בובות, מכוניות צעצוע, וצעצועי בעלי חיים. כל צעצוע מקבל מזהה ייחודי, שם ומחיר. עם הזמן, החנות התחילה למכור צעצועים חכמים המחוברים לאינטרנט, בעלי גרסת תוכנה וכוללים תכונות אינטראקטיביות, כמו שליטה מרחוק.

לדוגמה, צעצוע מסוג Smart Car הוא מכונית שאפשר לחבר לאפליקציה, ו-Smart Robot הוא רובוט שמסוגל להתחבר לאינטרנט ולתקשר עם הילד. עבור צעצוע חכם יש לשמור האם מחובר לאינטרנט (כן/לא) ואת גרסת התכנה המותקנת עליו.


```
public class SmartToy extends Toy
{
    private boolean isConnected;
    private String softwareVersion;
```

```
public SmartToy(String name, double price, String
softwareVersion)
{    super(name, price);
    this.softwareVersion = softwareVersion;
    this.isConnected = false;}
```

```
public boolean isConnected()
{ return isConnected;}
```

```
public void connect()
{ isConnected = true;}
```

```
public void disconnect()
{ isConnected = false;}
```

```
public String getSoftwareVersion()
{    return softwareVersion;}
```

```
public void updateSoftware(String newVersion)
{    this.softwareVersion = newVersion;
    System.out.println("Software updated to version: " +
newVersion);}
```

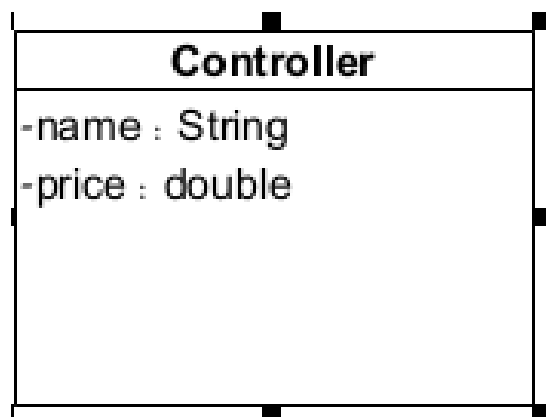
```
@Override
public void activateFeatures()
{    if (isConnected)
        System.out.println("Activating smart features for " + getName()
+ ": Online mode enabled.");
    else    System.out.println("Activating smart features for " +
getName() + ": Please connect to enable online features."); }
```

```
@Override public String toString()
{    return super.toString() + " - Smart Features [Connected: " +
isConnected + ", Software Version: " + softwareVersion + "]; }
} //end of class
```

המשך הסיפור

בעקבות הביקוש הגובר לצעצועים חכמים, החנות החלה לשווק בקרי שליטה שונים שמותאמים במיוחד לצעצועים החכמים.

בקרי השליטה, כמו Arduino ו-Raspberry Pi מאפשרים להפעיל פונקציות מתקדמות בצעצועים החכמים ולהפוך אותם לאינטראקטיביים. כל בקר מגיע עם מחיר משלו ושם הבקר.



```
public class Controller
```

```
{  
    private String name; // The name of the controller  
    private double price; // The price of the controller
```

```
  
    public Controller(String name, double price) {  
        this.name = name;  
        this.price = price;  
    }
```

```
  
    public String getName() {    return name;    }
```

```
  
    public void setName(String name) {    this.name = name;    }
```

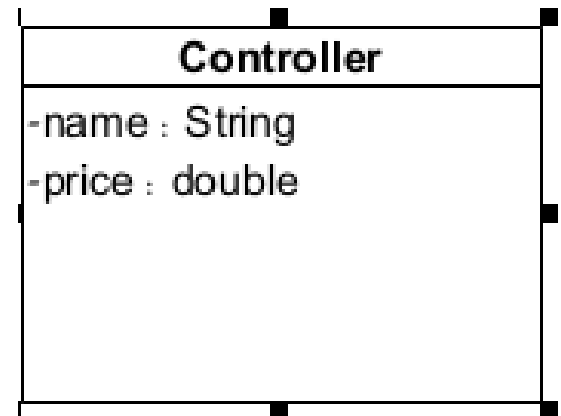
```
  
    public double getPrice() {    return price;    }
```

```
  
    public void setPrice(double price) {    this.price = price;    }
```

```
  
    @Override    public String toString()  
    {        return "Controller: " + name + " - Price: $" + price;    }  
}
```

עוד מחלקה

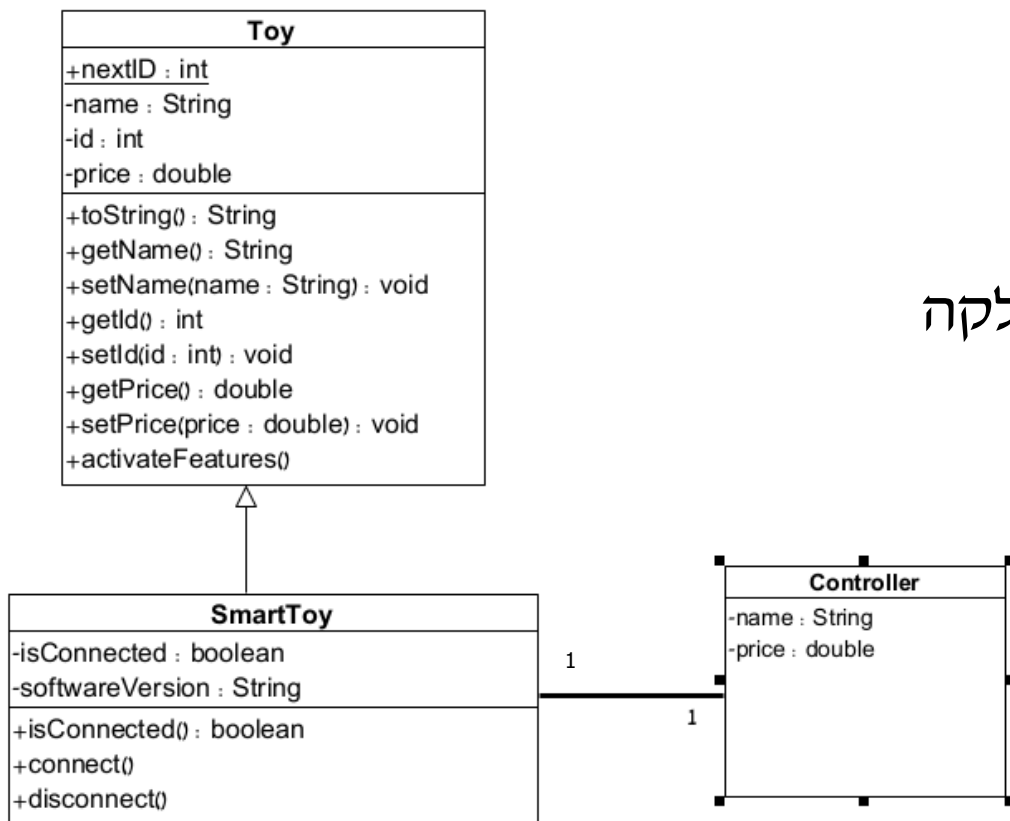
```
// Create some Controller objects  
Controller controller1 = new Controller("Arduino", 15.4);  
Controller controller2 = new Controller("RasberyPie", 134.5);
```



המשך הסיפור

לאחר שהחנות "Toys is us" החלה לשווק בקרי שליטה חכמים, הלקוחות גילו את היכולת לשדרג את הצעצועים החכמים שלהם ולהפוך אותם לאינטראקטיביים יותר. לקוח יכול לשדרג את הצעצוע החכם שקנה עם בקרים מתקדם כדי לאפשר יותר פונקציות משחק ולהעצים את חוויית המשחק. ברגע שמחברים את הבקר לצעצוע החכם, הצעצוע מתעדכן במחירו כך שיכלול את מחיר הבקר, ומאפשר גישה לתכונות של הבקר.

ישנה היכרות (קשר הכרות) בין מחלקת Controller ומחלקת Smart Toy - יבוא לידי ביטוי בקוד ע"י מחלקה מוכלת



```
public class SmartToy extends Toy
{
```

```
private Controller controller;
```

```
private boolean isConnected;
private String softwareVersion;
```

```
public SmartToy(String name, double price, Controller controller, String
softwareVersion)
```

```
{    super(name, price);
    this.controller = controller;
    this.softwareVersion = softwareVersion;
    this.isConnected = false;
updatePrice(); }
```

```
private void updatePrice()
```

```
{    if (controller != null)
    setPrice(super.getPrice() + controller.getPrice()); }
```

```
public Controller getController()
```

```
{    return controller; }
```

```
public void setController(Controller controller)
```

```
{    this.controller = controller;
    updatePrice(); }
```

```
public boolean isConnected() {    return isConnected; }
```

```
public void connect() {    isConnected = true; }
```

```
public void disconnect() {    isConnected = false; }
```

```
public String getSoftwareVersion(){return softwareVersion; }
```

```
public void updateSoftware(String newVersion)
```

```
{
```

```
    this.softwareVersion = newVersion;
```

```
    System.out.println("Software updated to version: " + newVersion); }
```

```
@Override    public void activateFeatures()
```

```
{    if (isConnected)
```

```
        System.out.println("Activating smart features for "
+ getName() + ": Online mode enabled with " +
controller.getName());
```

```
        else
```

```
        System.out.println("Activating smart features for "
+ getName() + ": Please connect to enable online
features with controller " + controller.getName()); }
```

```
@Override    public String toString()
```

```
{    return super.toString() + " - Smart Features
[Connected: " + isConnected + ", Software Version: " +
softwareVersion + "]" - " + controller.toString(); }
```

```
public class Main
{
    public static void main(String[] args)
    {
        // Create some Toy objects
        Toy toy1 = new Toy("Dog", 25.0);

        Controller controller1 = new Controller("Arduino", 15.4);
        Controller controller2 = new Controller("RasberyPie", 134.5);

        SmartToy smartToy1 = new SmartToy("Smart Car", 25.2, controller1);
        SmartToy smartToy2 = new SmartToy("Smart Robot", 40.5, controller2);

        // Create ArrayLists to hold the different objects
        ArrayList<Object> toys = new ArrayList<Object>();

        toys.add(toy1);
        toys.add(smartToy1);
        toys.add(smartToy2);
        toys.add(controller1);
        toys.add(controller2);
    }
}
```

```
System.out.println("All Objects:");
for (Object obj : toys)
    System.out.println(obj);

// Print only SmartToys
System.out.println("\nSmart Toys:");
for (Object obj : toys)
    if (obj instanceof SmartToy)
        System.out.println(obj);
}
} //end of class
```

All Objects:

```
Toy: Dog - ID: 1 - Price: $25.0
Smart Toy: Smart Car - ID: 2 - Price: $40.6 - Controller: Arduino - Price: $15.4
Smart Toy: Smart Robot - ID: 3 - Price: $175.0 - Controller: RasberryPie - Price: $134.5
Controller: Arduino - Price: $15.4
Controller: RasberryPie - Price: $134.5
```

Smart Toys:

```
Smart Toy: Smart Car - ID: 2 - Price: $40.6 - Controller: Arduino - Price: $15.4
Smart Toy: Smart Robot - ID: 3 - Price: $175.0 - Controller: RasberryPie - Price: $134.5
```

אב קדמון של כולם - Object



■ במחלקה Object מוגדות מתודות הבאות:

toString()
hashCode()
equals(Object obj)
finalize()
getClass()
clone()
wait(), notify() notifyAll()

■ בJava כל מחלקה יורשת ממחלקת Object אב קדמון

■ לכן כל המחלקות בהכרח דורסות מממשות או מקבלות את המתודות של מחלקת Object.

■ לכן כמעט בכל המחלקות שמממשת toString כתוב

@Override

דריסה overriding



- להבדיל מC# כל המתודות בJAVA מדומות virtual
 - ברירת מחדל
- לכן כל מתודה תמיד ניתנת לדריסה overriding.
- אם באמת עשינו דריסה למתודה במחקלה נגזרת אזי מקובל לשים מילה **@Override**

```
public class Toy  
{
```

```
    private static int nextId = 1;
```

```
    private String name;  
    private int id;  
    private double price;
```

```
    ...
```

```
    @Override public boolean equals(Object obj)  
    {
```

```
}
```

שאלה



יש להוסיף מתודה
?Toy **equals**
האם להתייחס ל?id

```
public class Toy
{
// Static variable to keep track of the next ID to be assigned
    private static int nextId = 1;
```

```
    private String name;
    private int id;
    private double price;
```

```
...
```

```
@Override
```

```
public boolean equals(Object obj)
```

```
{
```

```
    if (obj == null || !(obj instanceof Toy))
        return false;
```

```
    Toy other = (Toy)obj;
```

```
    return this.name.equals(other.getName()) && this.price == other.getPrice();
```

```
}
```

```
// Override the finalize method
```

```
@Override protected void finalize() throws Throwable
```

```
{ System.out.println("Finalizing " + this); }
```

יש להוסיף מתודות finalize ו-
Toy.equals

מחלקה אבסטרקטית



- מחלקה אבסטרקטית abstract מכילה מתודות אבסטרקטיות ויכולה להכיל מתודות רגילות.
- לא ניתן ליצור אובייקטים מסוג מחלקה זו.
- תחביר:

```
abstract class A
{
    public abstract void method();
    //other methods;
}
```

מחלקה מופשטת

```
public abstract class Man
{
    public abstract void sayHello();
    public void introduce() {
        System.out.println("I am a Man.");
    }
}
```

```
class Main
{
    public static void main(String[] args)
    {
        Man m = new Man (); //error
        m.introduce();
    }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Cannot instantiate the type Man1

    at Lecture1/Student.Main.main(Main.java:6)
```

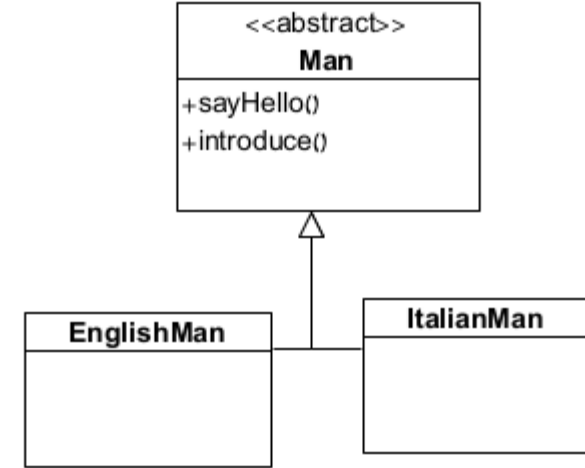
מחלקה מופשטת - 2

```
public abstract class Man
{
    public abstract void sayHello();
    public void introduce()
    {
        System.out.println("I am a Man.");
    }
}
```

```
public class EnglishMan extends Man
{
    public void sayHello()
    {
        System.out.println("Good morning!");
    }
}
```

```
public class ItalianMan extends Man
{
    public void sayHello()
    {
        System.out.println("Buongiorno!");
    }

    public void introduce()
    {
        System.out.println("I am an Italian Man.");
    }
}
```



```
class Main
{
```

```
    public static void main(String[] args)
    {
        //polymorphism
        Man im = new ItalianMan ();
        im.sayHello();
        im.introduce();
    }
```

```
<terminated> main [Java Application]
Buongiorno!
I am an Italian Man.
```

Final class



- מחלקה שלא ניתן לרשת ממנה
- final : Cannot be extended further

JAVA

```
public class Base
{
    protected void Foo() {}
}

public final class TypeResolver extends Base
{
    @Override
    protected final void Foo() {}
}
```

C#

```
public class Base
{
    protected virtual void Foo () {}
}

public sealed class TypeResolver : Base
{
    sealed protected override void Foo () {}
}
```

ממשק Interface



- ממשק זו מחלקה שכל המתודות שלה אבסטרקטיות.
- מקובל לתת שם שמתחיל מאות I
- תחביר:

```
interface IName {  
    public void method1(); // interface method  
    public void method2(); // does not have a body  
}
```

- יותר מאוחר מחלקה ממששת implements ממשק

```
class C implements IName{...}
```


המשך הסיפור

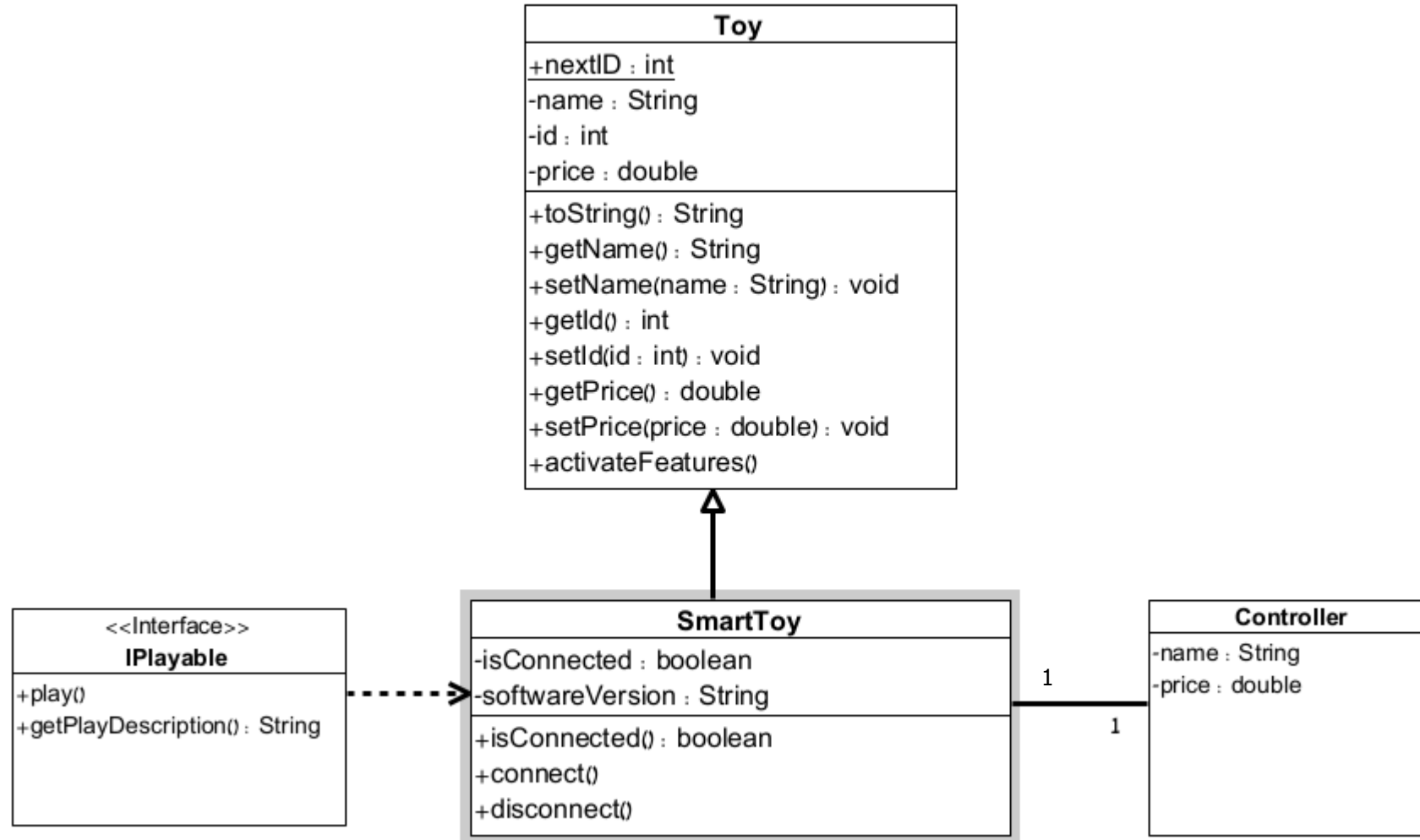
בעקבות ההצלחה של הצעצועים החכמים והבקרים, החנות החליטה להוסיף ממשק אינטראקטיבי חדש לצעצועים בשם `IPlayable`.

הממשק מאפשר לכל צעצוע חכם לתאר את חוויית המשחק שהוא מציע ולתת לילדים חוויה מתקדמת ואינטראקטיבית עוד יותר.

הממשק כולל שתי מתודות:

- `play` מפעילה את הצעצוע ומציגה את שם הצעצוע ואת שם הבקר, כך שהילדים יוכלו להבין איך הצעצוע נשלט.
 - `getPlayDescription` המחזירה תיאור קצר על חוויית המשחק שהצעצוע מציע.
- כאשר אובייקט `SmartToy` נוסף לרשימת הצעצועים של החנות, ניתן לבדוק האם הוא מממש את `IPlayable` ולהפעיל את מתודת `play` כך שהילד יוכל להתחיל לשחק מיד.
- התהליך הזה גם מאפשר לחנות להציג תיאורים שונים של הצעצועים בחנות – לפי סוג הצעצוע והבקר שהוא כולל.

דיאגרמת מחלקות



מימוש ממשיק

```
public interface IPlayable
{
    void play();
    String getPlayDescription();
}
```

```
public class SmartToy extends Toy implements IPlayable
```

```
{
    private Controller controller;
    private boolean isConnected;
    private String softwareVersion;
    ....
```

Like before....

.....

Like before....

//play מימוש מתודת IPlayable מהממשק

@Override

```
public void play()
```

```
{
```

```
    System.out.println("Playing with " + getName() +
        " using " + controller.getName() + ".");
```

```
}
```

//getPlayDescription מימוש מתודת IPlayable מהממשק

@Override

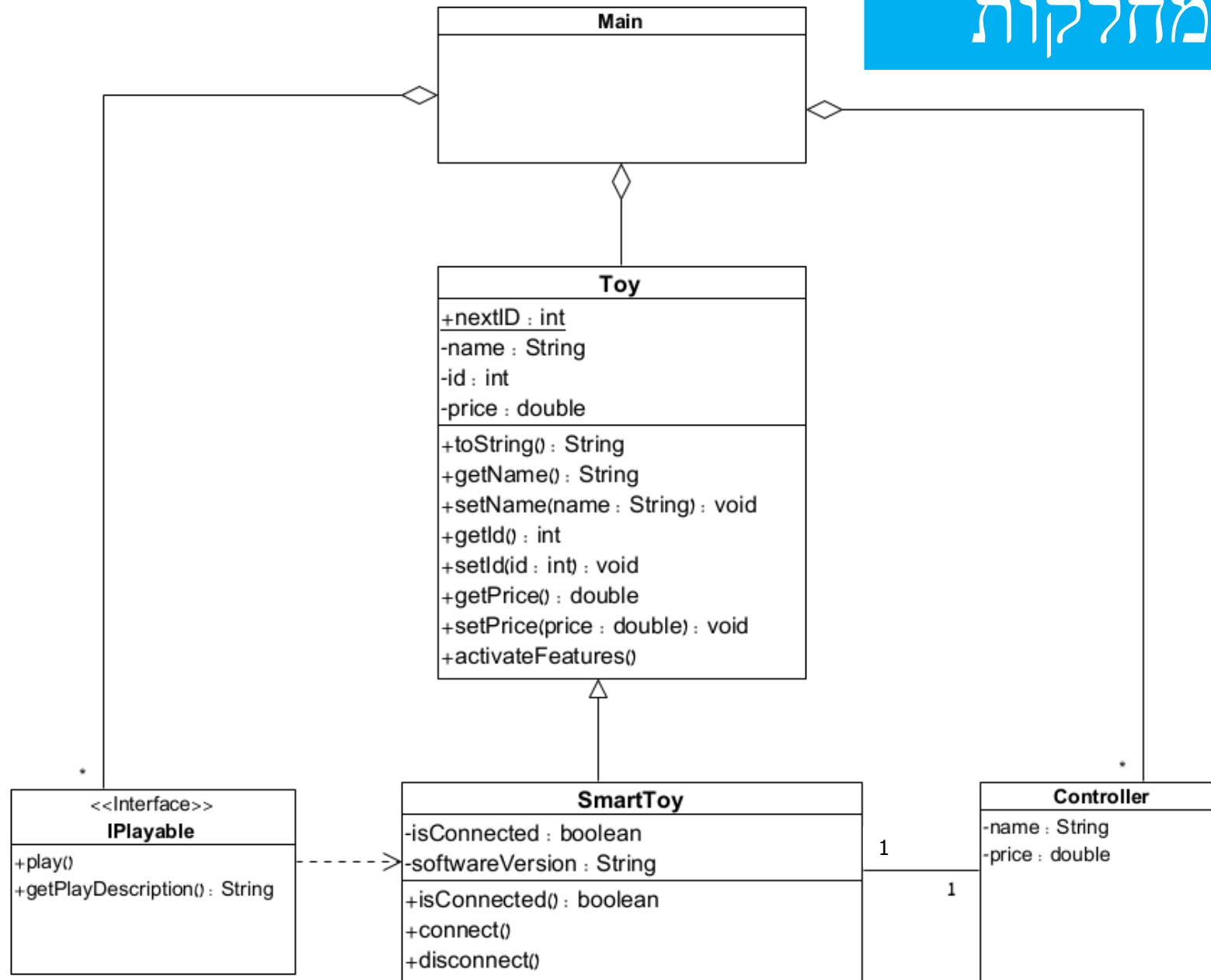
```
public String getPlayDescription()
```

```
{
```

```
    return "This smart toy is controlled by " +
        controller.getName() + ".";
```

```
}
```

דיאגרמת מחלקות



```
import java.util.ArrayList;

public class Main
{
    public static void main(String[] args)
    {
        // Create some Toy objects
        Toy toy1 = new Toy("Dog", 25.0);

        Controller controller1 = new Controller("Arduino", 15.4);
        Controller controller2 = new Controller("RasberyPie", 134.5);

        SmartToy smartToy1 = new SmartToy("Smart Car", 25.2, controller1);
        SmartToy smartToy2 = new SmartToy("Smart Robot", 40.5, controller2);

        // Create ArrayLists to hold the different objects
        ArrayList<Object> toys = new ArrayList<Object>();

        toys.add(toy1);
        toys.add(smartToy1);
        toys.add(smartToy2);
        toys.add(controller1);
        toys.add(controller2);
    }
}
```

```
// Print only SmartToys
System.out.println("\nSmart Toys:");
for (Object obj : toys)
    if (obj instanceof IPlayable)
        ((IPlayable) obj).play();
}
} //end of class
```

Smart Toys:

Playing with Smart Car using Arduino.

Playing with Smart Robot using Rasberrypie.

Instance_INITIALIZER

```
public class Example
{
    private final String _name ;
    private final int _type ;

    //instance initializer
    {
        _name = "Example" ;
    }

    //constructor
    public Example ( int type )
    { _type = type; }
}
```

- ניתן לבצע **איתחול מופע** לפני כניסה לבנאי.
- אין שם.
- בלוק {} ממקום בתוך המחלקה אבל אף לא שייד לאף מתודה או בנאי.
- לא ניתן לקרוא בצורה מפורשת.
- כל איתחולי המופעים מתרחשים מלמעלה כלפי מטה בקוד לפני קריאה לבנאי.
- חשוב: אם מדובר במחלקה נגזרת אז קודם מתבצעת קריאה לבנאי האב (super) ואז איתחולי מופע ורק לאחר מכן בנאי הבן.



סוגי חריגות



- ישנם חריגות מובנות/מוכנות (built-in) ויש חריגות משתמש.

- חריגות מובנות

- **נבדקות** *checked* חריגת זמן קומפילציה

- דוגמה IOException, SocketException

- **לא נבדקות** *unchecked* חריגת זמן הריצה

- דוגמה IndexOutOfBoundsException, ArithmeticException

- **חריגות משתמש** - חריגה שמשתמש יצר

```
class NewException extends Exception{ ...}
```


Throwable



- ב Java קיימת מחלקה Throwable שמייצגת מצב שגוי
- מחלקות Error ו- Exception יורשות ממנה.
- מנגנון:
- כאשר מתרחשת שגיאת ריצה נוצר אובייקט מסוג חריגה והוא "נזרק" `throw`
- בנוסף "דגל השגיאות" נדלק ומסמן שיש בעיה וכל עוד לא נטפל בה לא ניתן להמשיך בתוכנית הרגילה
- התוכנית יכולה "לתפוס" `catch` את החריגה ולטפל בה.

```
public class mainClass
{
    public static void main(String[] args)
    {
        try {
            // Create some Toy objects
            Toy toy1 = new Toy("Dog", 25.0);
            System.out.println(toy1);
            Toy toy2 = new Toy("Dog", -45.2);
            System.out.println(toy2);
        }
        catch (IllegalArgumentException e)
        {
            System.out.println("Exception caught: " + e.getMessage());
        }
    } //end of main
} //end of mainClass
```

```
Terminated: app [java / application] at [0:0:0] [para] [0:0:0] [comp] j
Toy: Dog - ID: 1 - Price: $25.0
Exception caught: Price cannot be negative.
```

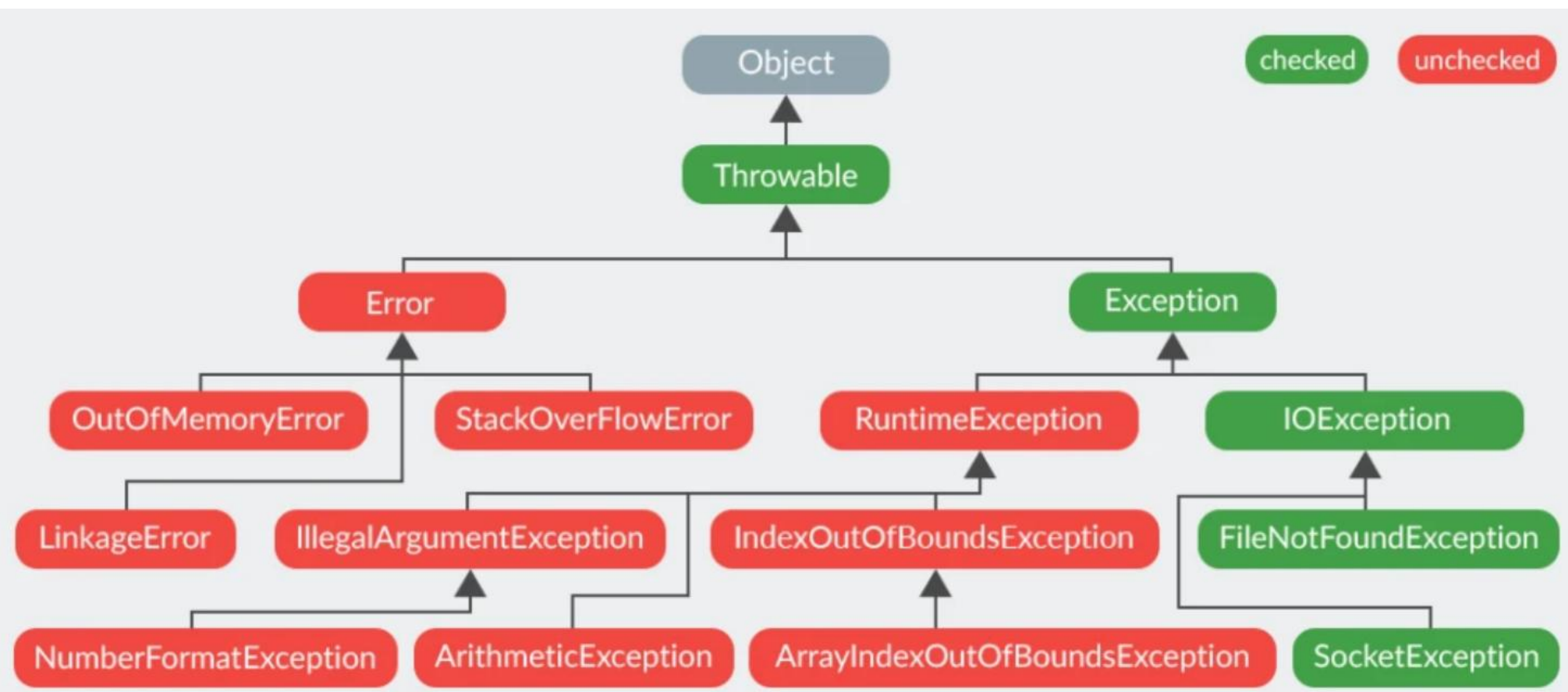
main

```
public class MainClass
{
    public static void main(String[] args)
    {

        // Create some Toy objects
        Toy toy1 = new Toy("Dog", 25.0);
        System.out.println(toy1);
        Toy toy2 = new Toy("Dog", -45.2);
        System.out.println(toy2);
    } //end of main
} //end of mainClass
```

כתיבה לא מסודרת
ללא try catch

```
<terminated> app [Java Application] C:\Users\para\Downloads\eclipse-java-2020-12-R-win32-x86_64\ec
Exception in thread "main" Toy: Dog - ID: 1 - Price: $25.0
java.lang.IllegalArgumentException: Price cannot be negative.
    at Toy.setPrice(Toy.java:82)
    at Toy.<init>(Toy.java:34)
    at app.main(app.java:14)
```



Checked Vs Unchecked

- כל שגיאה שיורשת מהמחלקות Error או RuntimeException מוגדרת **כלא** **נבדקות**.

- כל שאר החרیגות מוגדרות **כנבדקות**.

- חריגות נבדקות כל מתודה צריכה:

1. או לתפוס catch .. try את החריגה בגוף המתודה ולטפל בה

2. או להצהיר בחתימה שהיא לא תופסת את החריגה throws

- **דוגמה:**

```
public void fun() throws Exception1, Exception2
```

דוגמה חריגת משתמש

user exception

```
class InvalidPaymentException extends Exception
{
    public InvalidPaymentException(String message)
    {
        super(message);
    }
}
```

חריגות נפוצות



- NullPointerException -70% ■
- NumberFormatException – 55% ■
- IllegalArgumentException – 50% ■
- RuntimeException – 23% ■
- NoSuchMethodException – 16% ■
- ClassCastException – 15% ■
- Exception – 15% ■
- ArrayIndexOutOfBoundsException ■
- FileNotFoundException ■

חורשה וחרירות

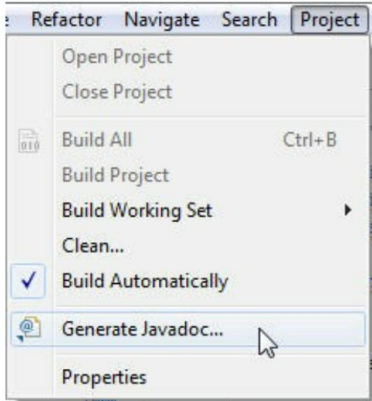
```
public class A
{
    public void fun() throws IOException,
        InterruptedException
        {... }
}

public class B extends A
{
    public void fun() throws EOFException,
        InterruptedException, ClassNotFoundException
        {... }
}
```

■ המתודה הדורסת **לא יכולה** להצהיר על חריגות שלא הוצהרו על ידי המתודה המקורית או שלא יורשות מחריגות כאלה

■ EOFException יורשת מ IOException ולכן **מותר** ClassNotFoundException לא מוגדרת אצל האב ולכן **אסור**

javadoc



■ פקודה ופורמט

■ התיעוד מיוצר מתוך הערות

■ javadoc עבור קובץ תקין מייצר קובץ HTML של המחלקה.

```
/** * Calculates the sum of two numbers.
```

```
*
```

```
* @param a the first number
```

```
* @param b the second number
```

```
* @return the sum of a and b
```

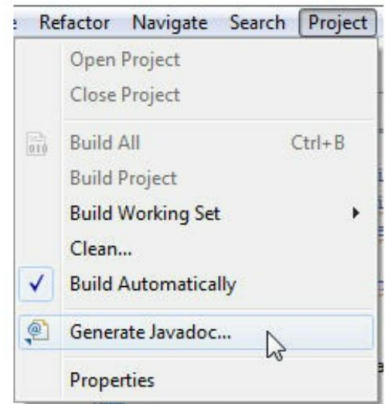
```
*/
```

```
public int sum(int a, int b)
```

```
{ return a + b; }
```

■ צורת כתיבה מתחיל ב /** / כל שורה * בסוף */.

■ דוגמה:



- search.js
- SmartToy.html
- stylesheet.css
- tag-search-index.js

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Class SmartToy

java.lang.Object
Toy
SmartToy

All Implemented Interfaces:
IPlayable

Version:
1.2

Author:
Rina Zviel

Constructor Summary

Constructors

```
/**
 * The {@code SmartToy} class represents a smart toy that inherits from Toy and includes an inner Controller object. It provide
 * and includes an inner Controller object. It provide
 * It provide
 * The price of the toy and the price of the controller.
 *
 * <p>This class is intended to be used in applications where managing smart toy and controller data is required.
 *
 * @author Rina Zviel
 * @version 1.2
 */
public class SmartToy extends Toy implements IPlayable {
    private Controller controller;

    /**
     * Constructor for SmartToy
     *
     * @param price The price of the toy
     * @param controller The controller object
     */
    public SmartToy(int price, Controller controller) {
        super(price);
        this.controller = controller;
    }

    /**
     * Get the price of the toy
     *
     * @return The price of the toy
     */
    public int getPrice() {
        return price;
    }

    /**
     * Get the controller object
     *
     * @return The controller object
     */
    public Controller getController() {
        return controller;
    }

    /**
     * Set the price of the toy
     *
     * @param price The price of the toy
     */
    public void setPrice(int price) {
        this.price = price;
    }

    /**
     * Set the controller object
     *
     * @param controller The controller object
     */
    public void setController(Controller controller) {
        this.controller = controller;
    }

    /**
     * Calculate the total price of the toy and the controller
     *
     * @return The total price
     */
    public int getTotalPrice() {
        return price + controller.getPrice();
    }

    /**
     * Play the toy
     *
     * @return The price of the toy
     */
    public int play() {
        return price;
    }
}
```

Questions

?

?

Answers

?