

# תוכנה 1 בשפת Java

## שיעור מספר 1: "שלום עולם"

### פרופ' ליאור וולף

בית הספר למדעי המחשב  
אוניברסיטת תל אביב



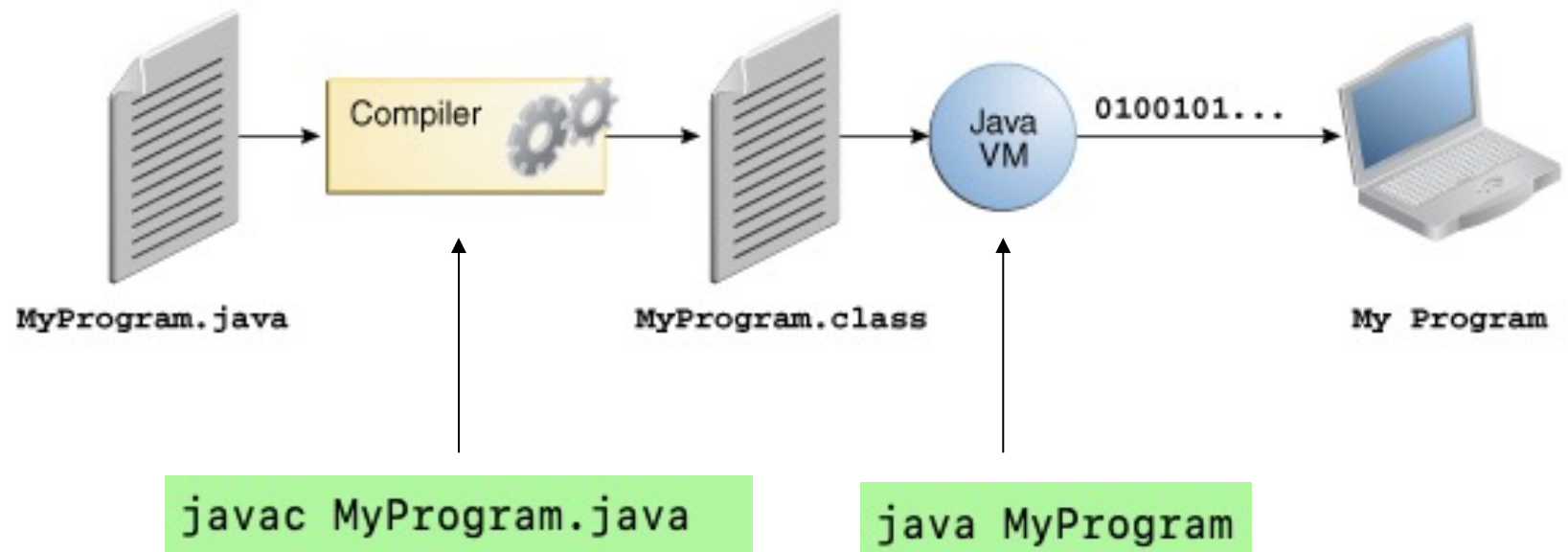
presented by kenichi-naitou.

# מה בתכנית?

- טעימה משפת Java
- פונקציית `main`
- 8 הטיפוסים היסודיים
- ביטויים ואופרטורים
- טיפוס המחרוזת וטיפוס המערך



# שלום עולם



<https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html>

# שלום עולם

Java Program

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

HelloWorldApp.java

Compiler



JVM



Win32

JVM



UNIX

JVM



MacOS

שלב הפיתוח

שלב הריצה

# המפרש (interpreter)

■ את הקוד שנכתב בשפת Java מריץ מפרש  
■ בדומה לשפת Python

■ לריצה בעזרת מפרש יש כמה חסרונות:

■ מאט את מהירות הריצה

■ טעויות מתגלות רק בזמן הריצה

■ לצורך כך הוסיפו ב Java שלב נוסף – הידור  
(compilation)

# המהדר (compiler)

- מבצע עיבוד מקדים של קוד התוכנית (שכתובה בקובץ טקסט רגיל) ויוצר קובץ חדש בפורמט **נוח יותר**
- קובץ זה אינו קריא למתכנת אנושי (אף שניתן לפתוח אותו בעורך טקסט כגון Notepad), אולם המבנה שלו מותאם לקריאה ע"י המפרש של Java
- פורמט זה נקרא byte code והוא נשמר בקובץ עם סיומת .class
- בתהליך העיבוד ("קומפילציה") נבדק התחביר של הקוד – והשגיאות המתגלות מדווחות למתכנת

# יבילות (portability)

- מדוע אנו מסתפקים בפורמט "נוח יותר"?
- מדוע אין המהדר יוצר קובץ בפורמט התואם בדיוק לחומרת המחשב, וכך היה נחסך בזמן ריצה גם שלב ה"הבנה" של הקוד?
- זאת מכיוון שאיננו יודעים מראש על איזה מחשב בדיוק תרוץ תוכנית ה-Java שכתבנו
- תוכניות Java חוצות סביבות (cross platform)
- סביבה = חומרה + מערכת הפעלה
- תוכנית שנכתבה והודרה במחשב מסוים, תוכל לרוץ בכל מחשב אשר מותקן בו מפרש ל-Java

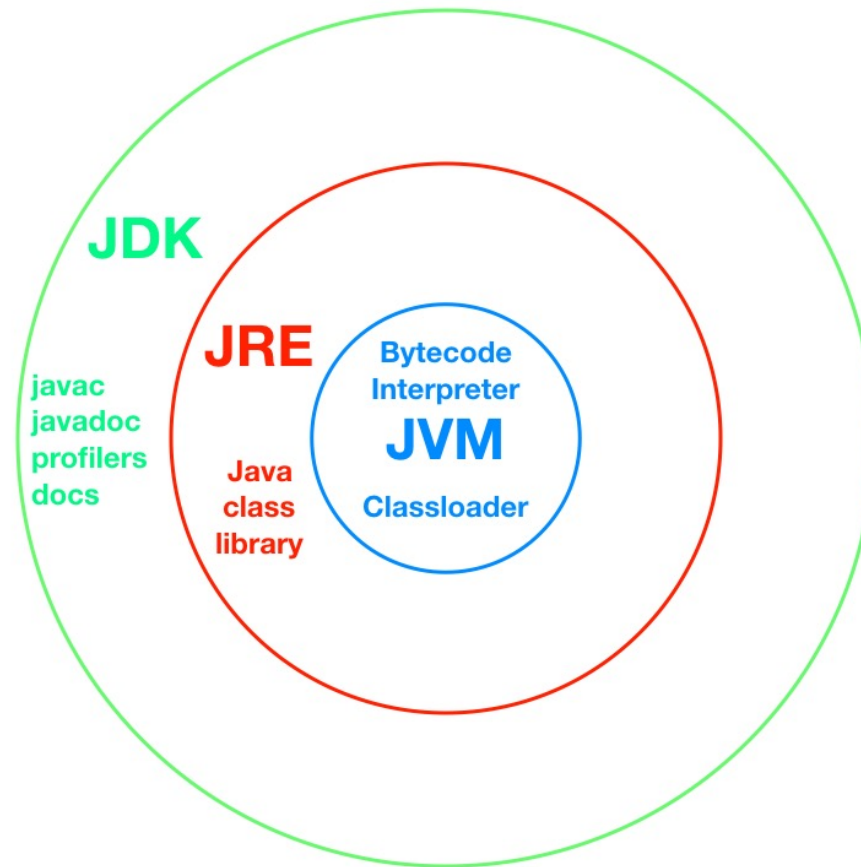
# המכונה המדומה

## (Java Virtual Machine)

- הקובץ המהודר מכיל הוראות ריצה ב"מחשב כללי" – הוא אינו עושה הנחות על ארכיטקטורת המעבד, מערכת ההפעלה, הזיכרון וכו'...
- עבור כל סביבה (פלטפורמה) נכתב מפרש מיוחד שיודע לבצע את התרגום מהמחשב הכללי, המדומה, למחשב המסוים שעליו מתבצעת הריצה
- את המפרש לא כותב המתכנת!
- דבר זה כבר נעשה ע"י ספקי תוכנה שזה תפקידם, עבור רוב סביבות הריצה הנפוצות



# איך הכל מתחבר?



<https://purelyfunctional.tv/guide/clojure-jvm/#abbreviations>



# שלום עולם

הגדרת מחלקה בשם HelloWorld.  
בשלב זה, נזהה מחלקה עם קובץ באותו שם

המחלקה ציבורית –  
ניתן להשתמש בה ללא  
הרשאות מיוחדות

חתימת המתודה

```
public class HelloWorld {
```

```
public static void main(String[] arguments) {
```

```
System.out.println("Hello World");
```

```
}
```

גוף המתודה

הגדרת מתודה (פונקציה)

יצירת תחום (scope)

# המתודה main

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

■ כאשר אנו מריצים מחלקה ה JVM מחפש מתודה עם

חתימה זו, ומריץ אותה

■ main – שם המתודה

■ public - המתודה ציבורית – ניתן להשתמש בה

ללא הרשאות מיוחדות

■ static – מתודה של המחלקה (יוסבר בהמשך)

■ void – טיפוס הערך המוחזר. למתודה זו אין ערך

מוחזר (ריק = void)

# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                               Boolean.parseBoolean(args[3]));  
    }  
}
```

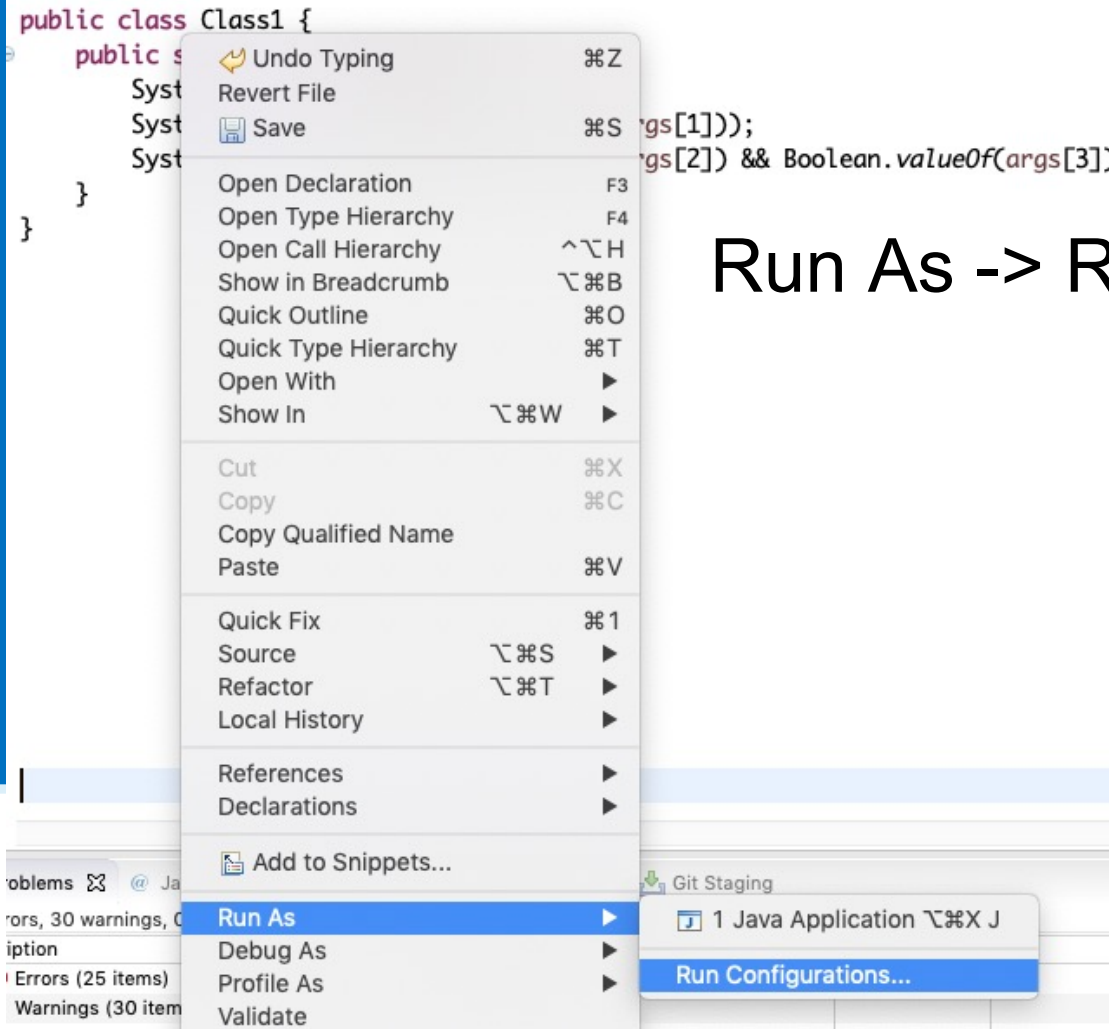
■ שם התוכנית: Class1.

■ התוכנית מבצעת 3 הדפסות. של מה?

■ פעולות שאנחנו מבצעים על הארגומנטים של התוכנית.

■ מאיפה הם מגיעים, הארגומנטים:

# תוכנית ראשונה



■ ב eclipse:

1. Run As -> Run Configuration

# תוכנית ראשונה

■ ב eclipse :

1. Run As -> Run Configuration

2. Arguments

run configurations



# תוכנית ראשונה

■ ב eclipse:

1. Run As -> Run Configuration

2. Arguments

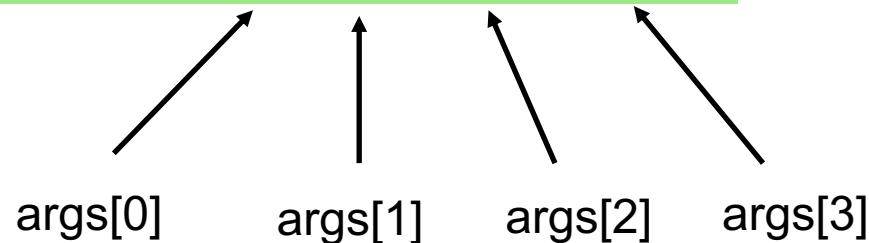
■ בשורת הפקודה:

```
java Class1 abc 12 true false
```

# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                               Boolean.parseBoolean(args[3]));  
    }  
}
```

java Class1 abc 12 true false

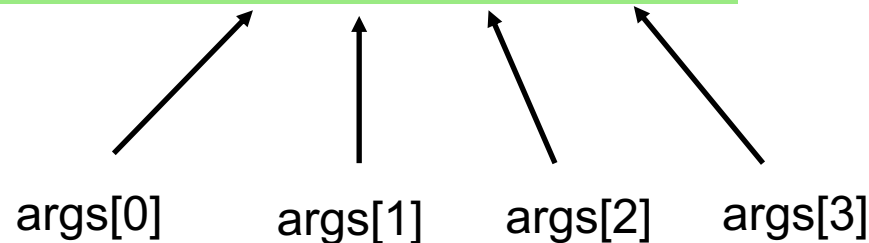




# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                               Boolean.parseBoolean(args[3]));  
    }  
}
```

```
java Class1 abc 12 true false
```

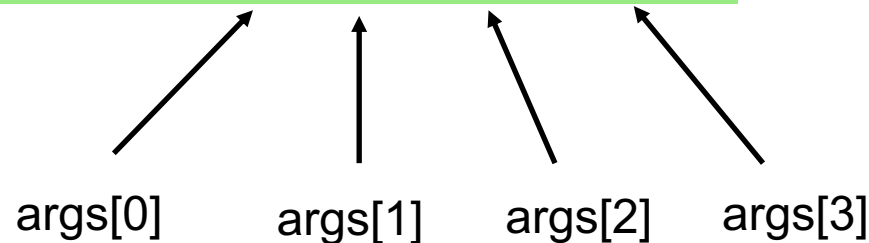


הדפסה של `args[0]` ללא שום פעולות נוספות

# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                               Boolean.parseBoolean(args[3]));  
    }  
}
```

```
java Class1 abc 12 true false
```

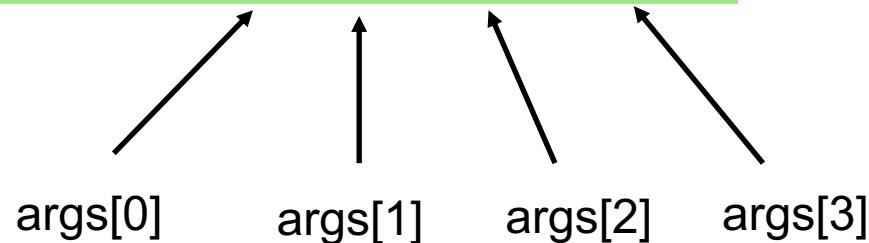


ממירים את `args[1]` מ `string` ל `int`, ומוסיפים לו 1.

# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                             Boolean.parseBoolean(args[3]));  
    }  
}
```

```
java Class1 abc 12 true false
```



ממירים את `args[2]` ואת `args[3]` מ `string` ל `boolean` ואז מחשבים את ערך פעולת ה `and` (וגם) עליהם.

# תוכנית ראשונה

```
public class Class1 {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(1 + Integer.parseInt(args[1]));  
        System.out.println(Boolean.parseBoolean(args[2]) &&  
                               Boolean.parseBoolean(args[3]));  
    }  
}
```

```
java Class1 abc 12 true false
```

```
abc  
13  
false
```

# מחרוזות – על קצה המזלג

```
public class Class1Strings {  
    public static void main(String[] args){  
        System.out.println(args[0]);  
        System.out.println(args[0].length());  
        System.out.println(args[0].charAt(0));  
        System.out.println(args[0].toUpperCase());  
    }  
}
```

```
java Class1Strings abc
```

```
abc  
3  
a  
ABC
```



---

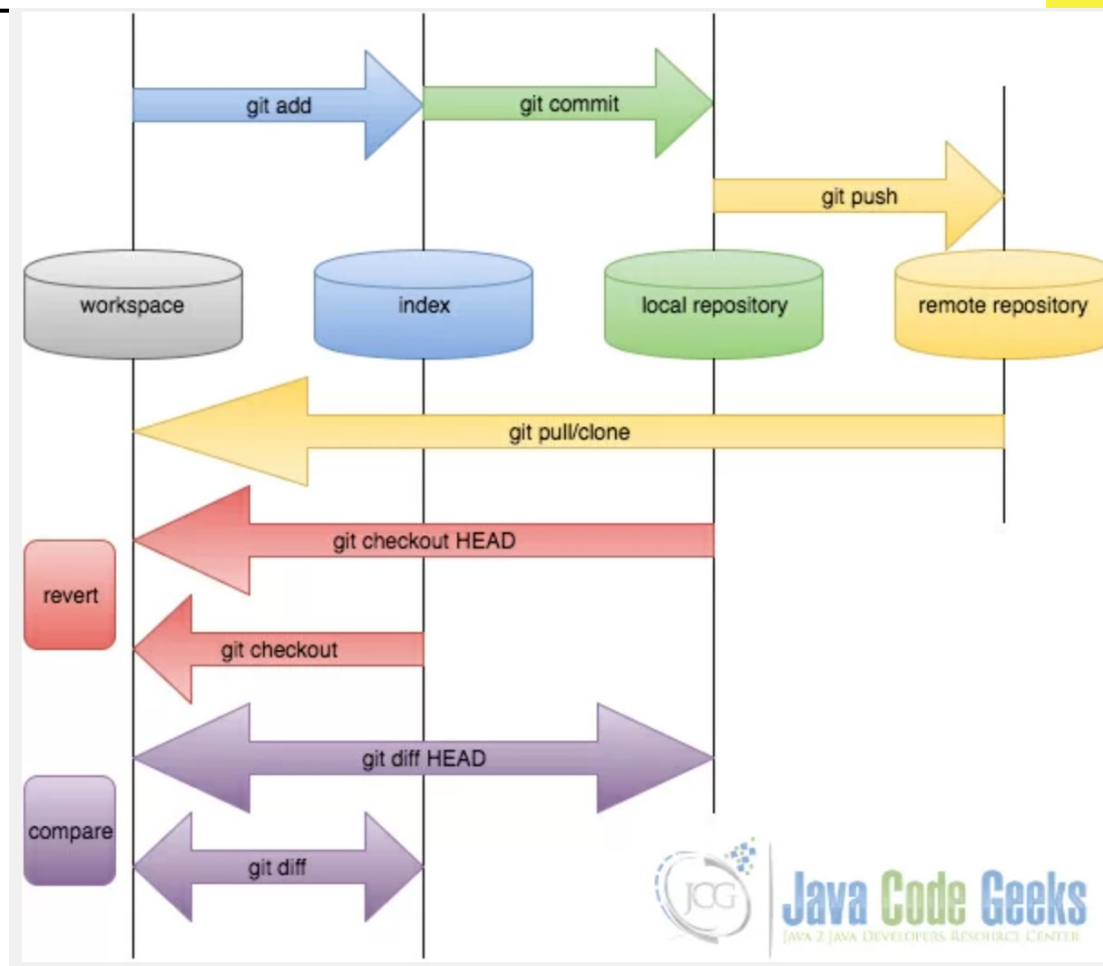
# Git

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# מה מאפשר לנו ה Git

- ניהול גירסאות של הקוד:
  - מעקב אחרי שינויים.
  - חזרה אחורה.
  - גיבוי.
- עבודה בצוות:
  - עדכון של שינויים שבוצעו ע"י חברי צוות אחרים.
  - מיזוג (אוטומטי או ידני) במידה ושני חברי צוות משנים את אותו הקוד.
- ככה עובדים היום בכל מקום (ואם זה לא Git, זה כלי בקרת תצורה אחר עם מאפיינים דומים)

# איך זה נראה?



<https://examples.javacodegeeks.com/software-development/git/git-tutorial-beginners/> מתוך:

תוכנה 1 בשפת Java  
אוניברסיטת תל אביב



# הסתעפויות

