תוכנה 1

תרגול 1: סביבת העבודה ומבוא ל-Java

ליאור שפירא ואורנית דרור

על הקורס

- קורס התכנות הבסיסי
- תכנות מונחה עצמים **באמצעות** בשפת Java
- הקורס ידרוש לימוד **עצמי** של פרטים
- אתר הקורס:

http://courses.cs.tau.ac.il/software1/0708a

- סביבת המחשוב באוניברסיטה היא Linux
  - תנאי קדם: פתיחת חשבון אישי במחשבי האוניברסיטה
  - הנחיות לפתיחת חשבון והכרת סביבת העבודה באתר הקורס.

סביבת פיתוח והרצה של Java

- גרסת ה-Java שעמה נעבוד:

Java SE (Standard Edition) 5.0

- חבילת סביבת ההרצה:

JRE (Java Runtime Environment) that includes:

- JVM (Java Virtual Machine)
- Standard Class Library

- חבילת ערכת הפיתוח:

JDK (Java Development Kit) that includes:

- JRE
- Command line tools: compiler, debugger etc.

- הורדה ותיעוד ב-http://java.sun.com/javase

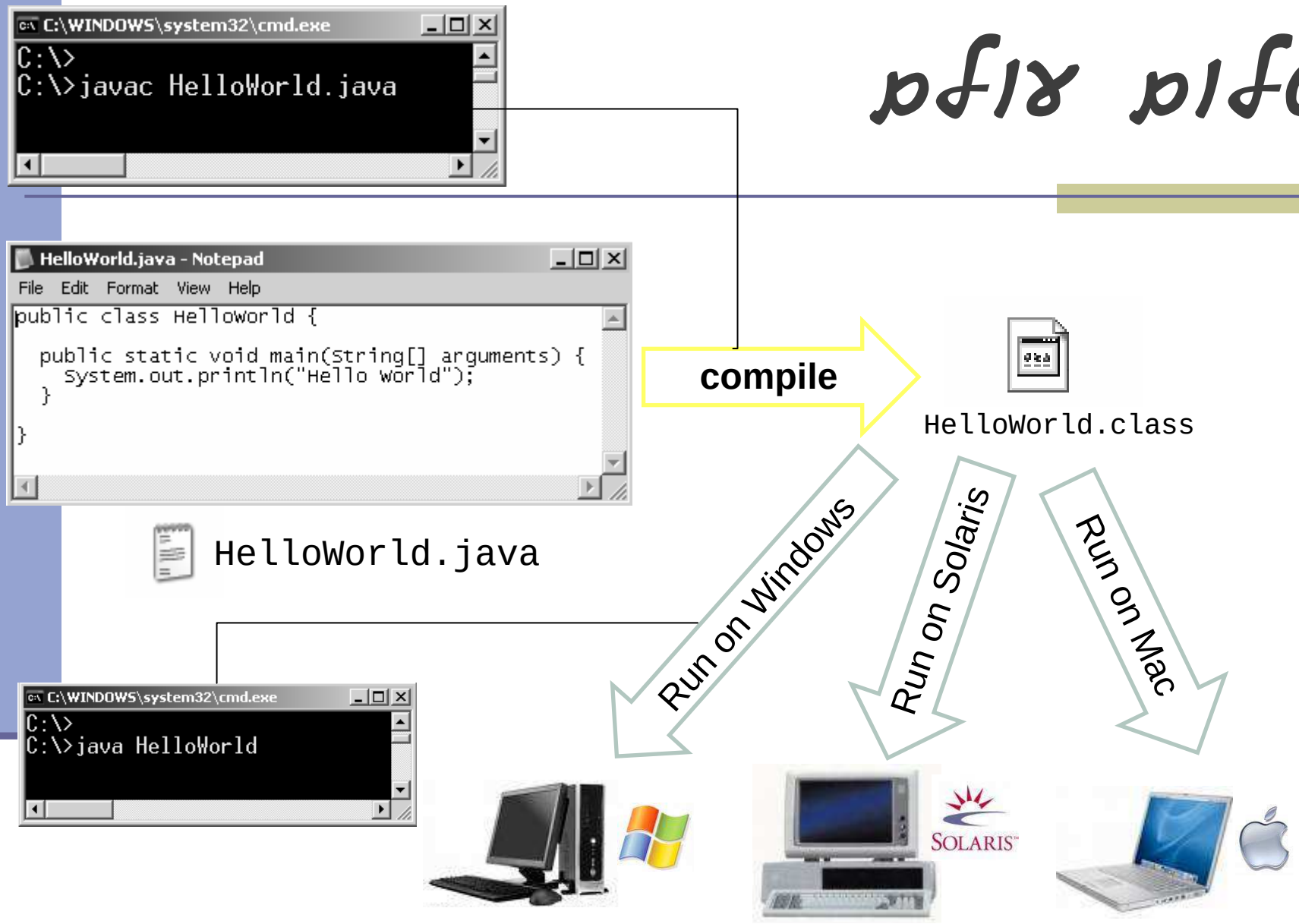
סביבת פיתוח שלובה

- **IDE** = Integrated Development Environment
- סביבה המשלבת רכיבי/כלי פיתוח עצמאיים:
  - עורך טקסט (editor)
  - סייר הקבצים (browser)
  - מהדר (compiler)
  - סביבת זמן ריצה (JRE)
  - מנפה השגיאות (debugger)
  - ועוד...
- **Eclipse** – ה- IDE בו נשתמש בקורס.

Eclipse

- IDE המתאים גם לפיתוח תוכנה ב Java
- ניתן להתקנה ב- Linux, Windows ועוד
- דורש התקנה בנפרד של JRE (או JDK)
- אתר הבית: www.eclipse.org
  - הורדת התכנה (גרסא 3.3.1) כקובץ zip (הוראות התקנה ב-הכרת סביבת המחשוב באתר הקורס)
  - אוסף גדול של מאמרים
- הכרות: דפי עבודה ללימוד Eclipse באתר הקורס
- דוגמא: פיתוח והרצת תכנית "Hello World" באקליפס

איך זה עובד



Write Once – Run Anywhere !

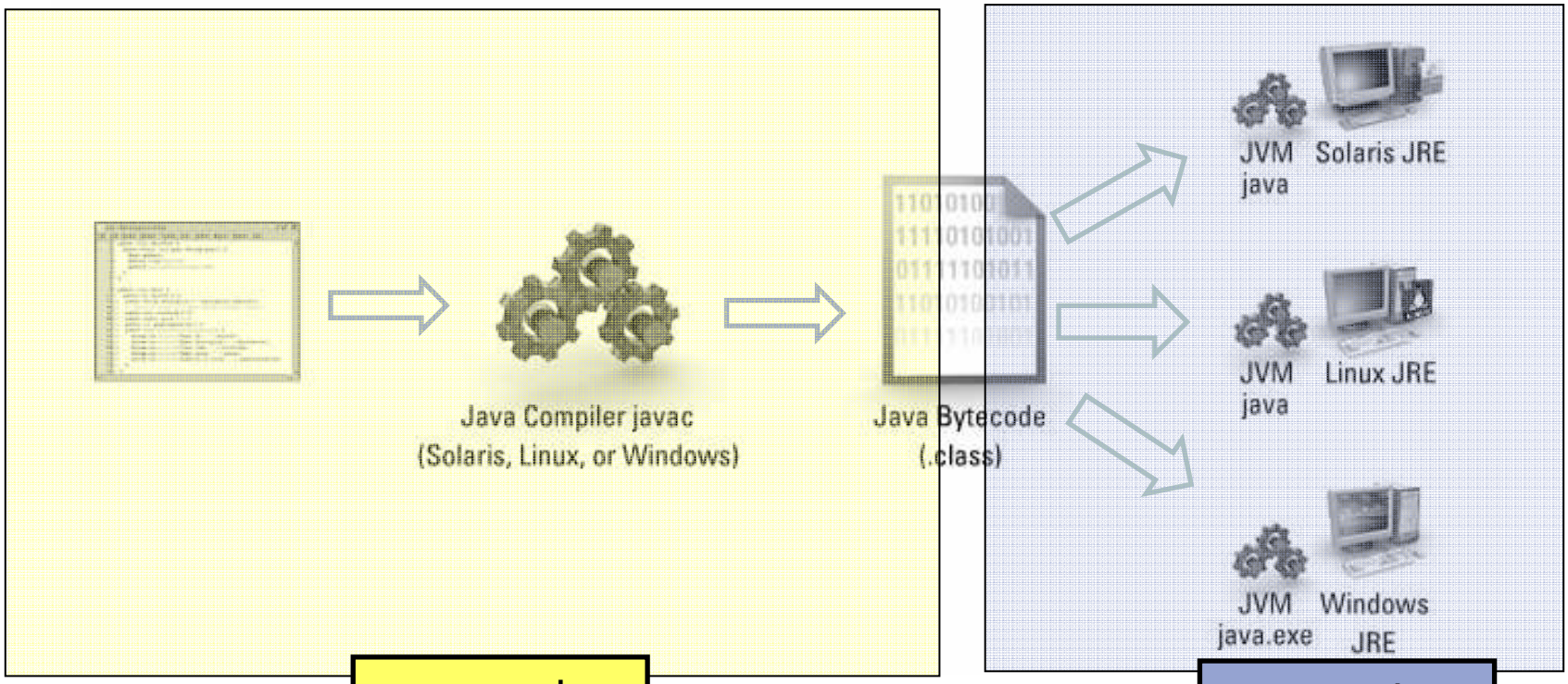
המפרש (Interpreter)

- המפרש מריץ את התכנית
  - בדומה ל-Scheme
- חסרונות הרצת תכנית ע"י מפרש:
  - מאט את מהירות הריצה
  - גילוי טעויות רק בזמן הריצה
- פתרון בשפת Java:
  - הוספת שלב נוסף – ***הידור*** (**compilation**)

המהדר (Compiler)

- מבצע עיבוד מקדים של קוד התכנית (קובץ ה-java.) ויוצר קובץ חדש בפורמט נוח יותר (קובץ ה-class.)
- בתהליך העיבוד ("**קומפילציה**") נבדק התחביר של הקוד ושגיאות מדווחות למתכנת
- במידה ואין שגיאות תחביר נוצר קובץ class.
  - פורמט הקובץ נקרא **byte code**
  - פורמט זה אינו קריא למתכנת אנושי (אף שניתן לפתוח אותו בעורך טקסט כגון Notepad)
  - הפורמט מותאם לקריאה ע"י המפרש של Java

עצמאות סביבתית
(Platform Independence)



9

שלום עולם

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

המחלקה ציבורית וניתנת לשימוש ללא כל מגבלה (הסבר בהמשך)

```
public class HelloWorld {

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

}
```

חתימת המתודה

גוף המתודה

הגדרת מתודה (פונקציה)

המתודה main

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

- אפשר להריץ רק מחלקות בעלות מתודה עם חתימה זו
  - main – שם המתודה
  - public - המתודה ציבורית ולכן ניתן להשתמש בה ללא הרשאות מיוחדות (יוסבר בהמשך)
  - static – מתודה של המחלקה (יוסבר בהמשך)
  - void – טיפוס הערך המוחזר. למתודה זו אין ערך מוחזר (ריק = void)

המתודה main

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

- String[] arguments – הגדרת משתנה פרמטר למתודה **בשם** arguments **ומטיפוס** מערך של מחרוזות
- לכל המשתנים ב-Java יש טיפוס המעיד על סוג וטווח הערכים שיכולים להיות מאוחסנים במשתנה (למשל: מחרוזת, מספר שלם, מספר ממשי, תו, מערך)
- שם המשתנה אינו חלק מהחתימה של המתודה

המתודה main

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

- System.out.println קריאה למתודת הדפסה
  - **זימון מתודה – method call**
  - השימוש כאן בשם המלא (qualified name) של המתודה המכיל את התו '.' (יוסבר בהמשך)
- העברת ארגומנט מטיפוס מחרוזת (String) – משפט עטוף במרכאות הוא מטיפוס מחרוזת
- משפטים ב Java מסתיימים בתו ';' (נקודה-פסיק)

טיפוסי השפה

- **טיפוסים יסודיים (פרימיטיביים)**: 8 טיפוסים מוגדרים בשפה שמיועדים להכיל ערכים פשוטים:
  - מספרים שלמים: byte, short, int, long
  - מספרים ממשיים: float, double
  - תווים: char
  - ערכים בוליאנים: boolean
- **טיפוסי הפנייה**: טיפוסים מורכבים היכולים גם להכיל מידע וגם לספק שרותים (יוסבר בהמשך)
  - המתכנת יכול להגדיר טיפוסי הפנייה חדשים
  - דוגמאות מיוחדות: מחרוזות ומערכים

הטיפוסים הפרימיטיביים

- בזכרון המחשב נשמר המידע בפורמט בינארי
- **סיבית** (bit): ספרה בינארית ('0' או '1')
- **בייט** (octet, byte): קבוצה של 8 סיביות

	Type	Size	Value Range
שלמים	`long`	64 bits	$-2^{63} \rightarrow 2^{63}-1$
	`int`	32 bits	$-2^{31} \rightarrow 2^{31}-1$
	`short`	16 bits	$-2^{15} \rightarrow 2^{15}-1$
	`byte`	8 bits	-128 →127
ממשיים	`double`	64 bits	beyond the scope of the discussion
	`float`	32 bits	
תווים	`char`	16 bits	most alphabet languages
ערכים לוגיים	`boolean`	“1 bit”	`true`, `false`

קשירת טיפוסים חזקה (Strong Typing)

- שלא כמו ב-Scheme:
  - יש להגדיר את הטיפוס של כל משתנה לפני השימוש בו

```
int number;
```

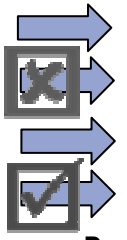
  - יש לציין בחתימת מתודה את הטיפוסים שעליהם היא פועלת

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

- טיפוס המשתנה מגדיר את הפעולות שניתן לבצע עליו
  - דוגמא: שנת לידה היא נתון מספרי שניתן לחסר לצורך חישוב גיל. שם הוא מחרוזת, שעליו אין הגיון לבצע חיסור
- הגדרת טיפוס לכל נתון מאפשרת זיהוי שגיאות בשלב הקומפילציה

הכרזה והשמה של משתנים מקומיים

```
public static void main(String[] args) {
    int i;
    System.out.println("i=" + i);
    i = 5;
    System.out.println("i=" + i);
}
```



אופרטור ה- '+' מבצע
שרשור מחרוזות כשלפחות
אחד מהאופרנדים הוא מחרוזת

- i הינו משתנה מקומי של המתודה
- **משפט הכרזה:** בזיכרון התכנית (באיזור שנקרא Stack, "המחסנית") מוקצים 4 בתים לצורך שמירת ערך עבור המשתנה i
- בנקודת זמן זו, הערך המופיע באזור זה חסר משמעות ("זבל")
- גישה לערך של i כעת היא טעות קומפילציה
- **משפט השמה:** סימן ה'=' אינו מציין השוואה אלא השמה (בדומה ל set! ב-Scheme). לכן, הערך 5 ייכתב לתוך הזיכרון שהוקצה למשתנה i
- כעת, הגישה למשתנה i תקינה ויודפס למסך: i=5

השמה

- תחביר ההשמה הוא:

<variable> = <expression>

- השמה מתבצעת "מימין לשמאל":
  1. מחושבת תוצאת הביטוי שבאגף ימין
  2. ערך הביטוי נכתב לתוך המשתנה שבאגף שמאל
- דוגמא:

```
int i;
i = 1 + 2      // i gets the value 3
```

השמה

- מה קורה אם הביטוי באגף ימין הוא משתנה בעצמו?

```
int x = 5;
int y = x;
```

- מה יקרה ל- y אם נשנה עכשיו את x?

```
x = 3;
```

- מה יקרה ל- x אם נשנה עכשיו את y?

```
y = 7;
```

- בהשמה **מועתקת** תוצאת חישוב הביטוי שבאגף ימין

אתחול משתנים מקומיים

- ניתן לשלב את ההצהרה וההשמה ע"י משפט אתחול:

```
int i = 7;
int j = 2 + 3;
```

- במשפט אתחול המשתנה נוצר (הוקצה לו זכרון) והושם לו ערך באותה הפעולה
- כך מובטח כי הגישה למשתנה תהיה תמיד בטוחה
- אין ב-Java אתחול ברירת מחדל למשתנים מקומיים