תוכנה 1 בשפת Java

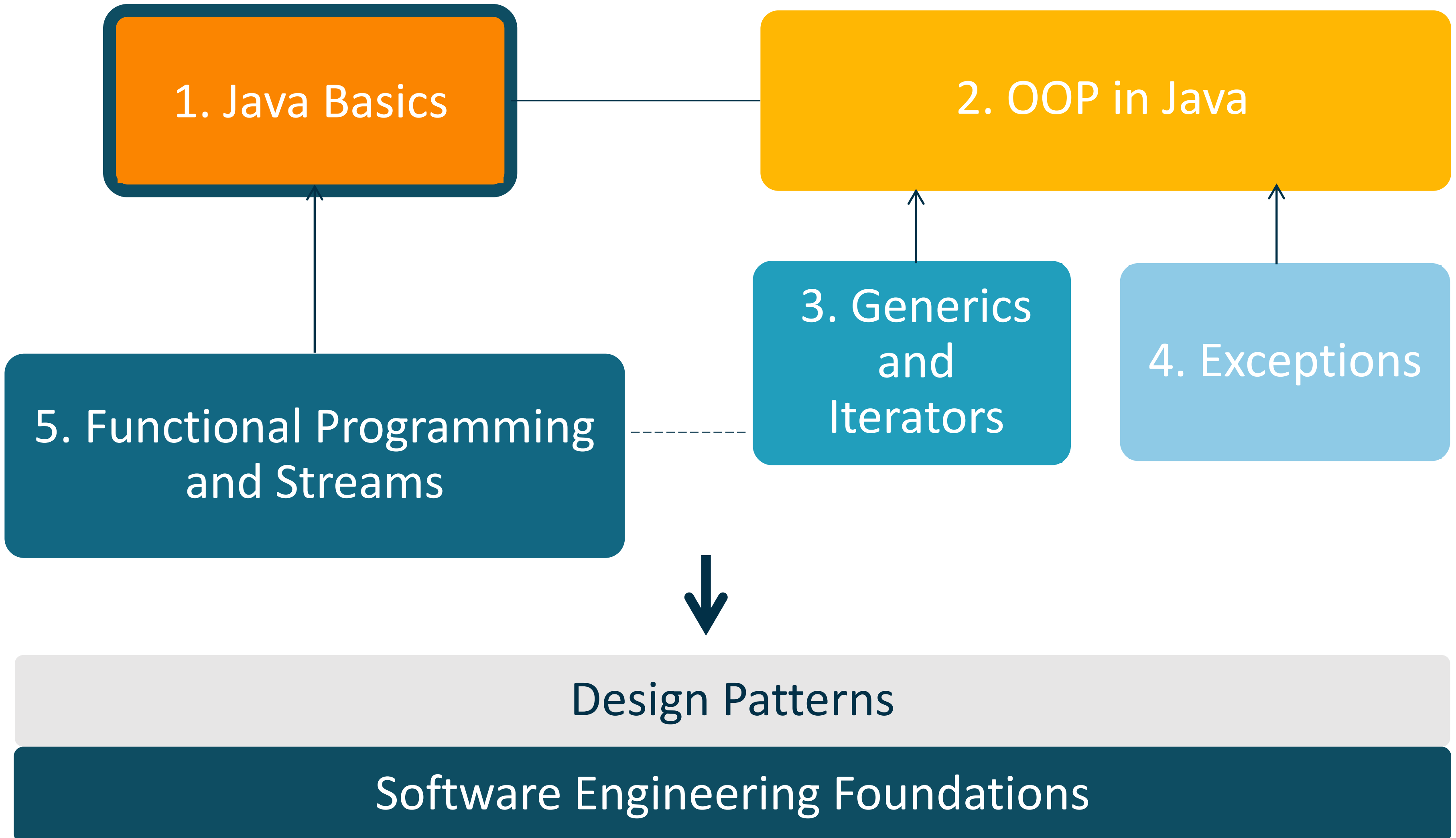
שיעור מספר 2: מבני בקרה, העמסה, מודל הזיכרון

מיכל קליינבורט

בית הספר למדעי המחשב

אוניברסיטת תל אביב

נושאי הקורס



התוכנית להיום

תכנות בסיסי ב Java - בעבודה עצמית 2:
if/else, לולאות (for, while, do-while), לולאות על מערכים
ולולאת for each

בלוק switch case

שירותי מחלקה, משתני מחלקה

העמסה

מודל הזיכרון של Java:
Heap and Stack, העברת ארגומנטים

התוכנית להיום

תכנות בסיסי ב Java - בעבודה עצמית 2:
if/else, לולאות (for, while, do-while), לולאות על מערכים ולולאת for each

בלוק switch case

שירותי מחלקה, משתני מחלקה

העמסה

מודל הזיכרון של Java:
Heap and Stack, העברת ארגומנטים

ריבוי תנאים (switch)

- קיים תחביר מיוחד לריבוי תנאים:
- ארגומנט ה **switch** יכול להיות byte, short, int, char, String או טיפוס מניה (יוסבר בהמשך הקורס)
- מתבצעת השוואה בינו ובין כל אחד מערכי ה **case** ומתבצעת קפיצה לשורה המתאימה אם קיימת
- לאחר הקפיצה מתחיל ביצוע סדרתי של המשך התוכנית, תוך התעלמות משורות ה **case**
- מה יודפס עבור grade שהוא 60?
- מה יודפס עבור grade שהוא 70?

```
System.out.print("Your grade is: ");

switch(grade){
     case 100:
       System.out.println("A+");
     case 90:
       System.out.println("A");
     case 80
       System.out.println("B");
     case 70:
       System.out.println("C");
     case 60:
       System.out.println("D");
}
...
```

ריבוי תנאים (break)

```java
System.out.print("Your grade is: ");

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;
}
...
```

- ניתן לסיים משפט **switch** לאחר ההתאמה הראשונה, ע"י המבנה **break**
- מה יודפס עבור grade שהוא 70?
- מה יודפס עבור grade שהוא 50?
- שימוש במבנה **default** (ברירת מחדל) נותן מענה לכל הערכים שלא הופיעו ב **case** משלהם
- מקובל למקם את ה **default** כאפשרות האחרונה

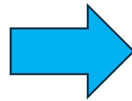
ריבוי תנאים (default)

```
System.out.print("Your grade is: ");

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");
}
...
```

- מה יודפס עבור grade שהוא 50?
- בתכנות מונחה עצמים נשתדל להימנע משימוש ב switch

ריבוי תנאים

```java
int numLetters = 0;
String day = args[0];
switch (day) {
      case "MONDAY":
      case "FRIDAY":
      case "SUNDAY":
            numLetters = 6;
            break;
      case "TUESDAY":
            numLetters = 7;
            break;
      case "THURSDAY":
      case "SATURDAY":
            numLetters = 8;
            break;
      case "WEDNESDAY":
            numLetters = 9;
            break;
      default:
            System.err.println("Illegal day "
                  + day);
}
System.out.println(numLetters);
```



```java
int numLetters = 0;
String day = args[0];
switch (day) {
      case "MONDAY", "FRIDAY", "SUNDAY":
            numLetters = 6;
            break;
      case "TUESDAY":
            numLetters = 7;
            break;
      case "THURSDAY", "SATURDAY":
            numLetters = 8;
            break;
      case "WEDNESDAY":
            numLetters = 9;
            break;
      default:
            System.err.println("Illegal  day " + day);
}
System.out.println(numLetters);
```

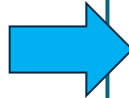
ריבוי תנאים (switch)

- המבנה הנ"ל נקרא switch statement
- החל מ Java 15 התווסף לשפה המבנה switch expression
  - במבנה זה בלוק ה switch יכול גם להחזיר ערכים

Switch statement vs. expression

Switch Statement

```
int numLetters = 0;
String day = args[0];
switch (day) {
    case "MONDAY", "FRIDAY", "SUNDAY":
        numLetters = 6;
        break;
    case "TUESDAY":
        numLetters = 7;
        break;
    case "THURSDAY", "SATURDAY":
        numLetters = 8;
        break;
    case "WEDNESDAY":
        numLetters = 9;
        break;
    default:
        System.err.println("Illegal  day "
                        + day);
}
System.out.println(numLetters);
```

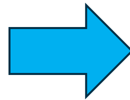


Switch Expression

```
String day = args[0];
int numLetters = switch (day) {
    case "MONDAY", "FRIDAY", "SUNDAY":{
        yield 6;
    }
    case "TUESDAY": {
        yield 7;
    }
    case "THURSDAY", "SATURDAY":{
        yield 8;
    }
    case "WEDNESDAY":{
        yield 9;
    }
    default:{
        System.err.println("Illegal day " +
                day);
        yield 0;
    }
};
System.out.println(numLetters);
```

Switch expression

```java
String day = args[0];
int numLetters = switch (day) {
    case "MONDAY", "FRIDAY", "SUNDAY":{
        yield 6;
    }
    case "TUESDAY": {
        yield 7;
    }
    case "THURSDAY", "SATURDAY":{
        yield 8;
    }
    case "WEDNESDAY":{
        yield 9;
    }
    default:{
        System.err.println("Illegal day "
                        + day);
        yield 0;
    }
};
System.out.println(numLetters);
```



```java
String day = args[0];
int numLetters = switch (day) {
    case "MONDAY", "FRIDAY", "SUNDAY" -> 6;
    case "TUESDAY" -> 7;
    case "THURSDAY", "SATURDAY" -> 8;
    case "WEDNESDAY" -> 9;
    default -> {
        System.err.println("Illegal day " +  day);
        yield 0;
    }
};
System.out.println(numLetters);
```

התוכנית להיום

תכנות בסיסי ב Java - בעבודה עצמית 2:
if/else, לולאות (for, while, do-while), לולאות על מערכים ולולאת for each

בלוק switch case

שירותי מחלקה, משתני מחלקה

העמסה

מודל הזיכרון של Java:
Heap and Stack, העברת ארגומנטים

שרותי מחלקה (static methods)

- בדומה לשיגרה (פרוצדורה, פונקציה) בשפות תכנות אחרות, *שרות מחלקה* הוא סדרת משפטים שניתן להפעילה ממקום אחר בקוד של ג'אווה ע"י קריאה לשרות, והעברת אפס או יותר ארגומנטים

- שירותים כאלה מוכרזים על ידי מילת המפתח **static** כמו למשל:

```
public class MethodExamples {

  public static void printMultipleTimes(String text, int times) {
      for(int i=0; i<times; i++)
          System.out.println(text);
  }

}
```

- נתעלם כרגע מההכרזה **public**

הגדרת שרות

- התחביר של הגדרת שרות הוא:

```
<modifiers> <type> <method-name> ( <paramlist> ) {
  <statements>
}
```

- <modifiers> הם 0 או יותר מילות מפתח מופרדות ברווחים (למשל public static)
- <type> מציין את טיפוס הערך שהשרות מחזיר
  - **void** מציין שהשרות אינו מחזיר ערך
- <paramlist> רשימת הפרמטרים הפורמליים, מופרדים בפסיק, כל אחד מורכב **מטיפוס** הפרמטר **ושמו**

החזרת ערך משרות ומשפט return

- משפט **return**:

```
return <optional-expression>;
```

- ביצוע משפט **return** מחשב את הביטוי (אם הופיע), מסיים את השרות המתבצע ברגע וחוזר לנקודת הקריאה

- אם המשפט כולל ביטוי ערך מוחזר, ערכו הוא הערך שהקריאה לשרות תחזיר לקורא

- טיפוס הביטוי צריך להיות תואם לטיפוס הערך המוחזר של השרות

- אם טיפוס הערך המוחזר מהשרות הוא **void** , משפט ה- **return** לא יכלול ביטוי, או שלא יופיע משפט **return** כלל, והשרות יסתיים כאשר הביצוע יגיע לסופו

דוגמה לשרות המחזיר ערך

```
public static int arraySum(int [] arr) {
   int sum = 0;
   for (int i : arr) {
      sum += i;
   }
   return sum;
}
```

- אם שרות מחזיר ערך, כל המסלולים האפשריים של אותו שרות (`flows`) צריכים להחזיר ערך
- איך נתקן את השרות הבא:

```
public static int returnZero() {
   int one = 1;
   int two = 2;

   if (two > one)
      return 0;
}
```

גוף השרות

- גוף השרות מכיל הצהרות על משתנים מקומיים (variable declaration) ופסוקים ברי ביצוע (כולל return)
- משתנים מקומיים נקראים גם משתנים זמניים, משתני מחסנית או משתנים אוטומטיים
- הצהרות יכולות להכיל פסוק איתחול בר ביצוע (ולא רק אתחול ע"י ליטרלים)

```
public static void doSomething(String str) {
  int length = str.length();
  ...
```

- הגדרת משתנה זמני צריכה להקדים את השימוש בו
- תחום הקיום של המשתנה הוא גוף השרות
- חייבים לאתחל או לשים ערך באופן מפורש במשתנה לפני השימוש בו

יש צורך באתחול מפורש

- קטע הקוד הבא, לדוגמה, אינו עובר קומפילציה:

```
int i;
int one = 1;

if (one == 1) // what about if (true)?
  i = 0;

System.out.println("i=" + i);
```

הקומפיילר צועק ש- i עלול שלא להיות מאותחל לפני השימוש בו

קריאה לשרות (method call)

- קריאה לשרות (לפעמים מכונה – "זימון מתודה") שאינו מחזיר ערך (טיפוס הערך המוחזר הוא void) תופיע בתור משפט (פקודה), ע"י ציון שמו וסוגריים עם או בלי ארגומנטים

- קריאה לשרות שמחזיר ערך תופיע בדרך כלל כביטוי (למשל בצד ימין של השמה, כחלק מביטוי גדול יותר, או כארגומנט המועבר בקריאה אחרת לשרות)

- קריאה לשרות שמחזיר ערך יכולה להופיע בתור משפט (statement), אבל יש בזה טעם רק אם לשרות תוצאי לוואי, כי הערך המוחזר הולך לאיבוד

- גם אם השרות אינו מקבל ארגומנטים, יש חובה לציין את הסוגריים אחרי שם השרות

```java
public class MethodCallExamples {

    public static void printMultipleTimes(String text, int times) {
        for (int i = 0; i < times; i++)
            System.out.println(text);
    }

    public static int arraySum(int[] arr) {
        int sum = 0;
        for (int i : arr)
            sum += i;
        return sum;
    }
    public static void main(String[] args) {

        printMultipleTimes("Hello", 5);

        int[] primes = { 2, 3, 5, 7, 11 };
        int sumOfPrimes = arraySum(primes);
        System.out.println("Sum of primes is: " + sumOfPrimes);
    }
}
```

הגדרת שרות void (פרוצדורה)

הגדרת שרות עם ערך מוחזר (פונקציה)

קריאה לשרות

קריאה לשרות

```java
public class MethodCallExamples {

    public static void printMultipleTimes(String text, int times) {
        for (int i = 0; i < times; i++)
          System.out.println(text);
    }

    public static int arraySum(int[] arr) {
        int sum = 0;
        for (int i : arr)
          sum += i;
        return sum;
    }

    public static void main(String[] args) {

        printMultipleTimes("Hello", 5);

        int[] primes = { 2, 3, 5, 7, 11 };
        int sumOfPrimes = arraySum(primes);
        System.out.println("Sum of primes is: " + sumOfPrimes);
        System.out.println("Sum of primes is: " + arraySum(primes));
    }
}
```

ניתן לוותר על משתנה העזר sumOfPrimes

```java
public class MethodCallExamples {

    public static void printMultipleTimes(String text, int times) {
        for (int i = 0; i < times; i++)
          System.out.println(text);
    }

    public static int arraySum(int[] arr) {
        int sum = 0;
        for (int i : arr)
          sum += i;
        return sum;
    }

    public static void main(String[] args) {

        printMultipleTimes("Hello", 5);

        int[] primes = { 2, 3, 5, 7, 11 };
        arraySum(primes);
    }
}
```

אין חובה לקלוט את הערך
המוחזר משרות,
אולם אז הוא הולך לאיבוד

שם מלא (qualified name)

- אם אנו רוצים לקרוא לשרות מתוך שרות של מחלקה אחרת (למשל main), יש להשתמש בשמו המלא של השרות

- שם מלא כולל את שם המחלקה שבה הוגדר השרות ואחריו נקודה

```
public class CallingFromAnotherClass {

    public static void main(String[] args) {
        printMultipleTimes("Hello", 5);

        int[] primes = { 2, 3, 5, 7, 11 };
        System.out.println("Sum of primes is: " +
            arraySum(primes));
    }
}
```

שם מלא (qualified name)

- במחלקה המגדירה ניתן להשתמש בשם המלא של השרות, או במזהה הפונקציה בלבד (unqualified name)
- בצורה זו ניתן להגדיר במחלקות שונות פונקציות עם אותו השם (מכיוון שהשם המלא שלהן שונה אין התלבטות – no ambiguity)

```
public class CallingFromAnotherClass {

    public static void main(String[] args) {
        MethodCallExamples.printMultipleTimes("Hello", 5);

        int[] primes = { 2, 3, 5, 7, 11 };
        System.out.println("Sum of primes is: " +
                           MethodCallExamples.arraySum(primes));
        }
}
```

- כבר ראינו **שימוש אחר** באופרטור הנקודה כדי לבקש בקשה מעצם, זהו שימוש נפרד שאינו שייך להקשר זה

משתני מחלקה

- עד כה ראינו משתנים מקומיים – משתנים זמניים המוגדרים בתוך מתודה, בכל קריאה למתודה הם נוצרים וביציאה ממנה הם נהרסים

- ב Java ניתן גם להגדיר **משתנים גלובליים** (global variables)
  - מכונים שדות סטטיים (static fields/members)

- משתנים אלו יוגדרו בתוך גוף המחלקה אך מחוץ לגוף של מתודה כלשהי, ויסומנו ע"י המציין **`static`**.
  - יכולה להיות ניראות שונה (public/private)

משתני מחלקה לעומת משתנים מקומיים

- משתנים אלו, שונים ממשתנים מקומיים בכמה מאפיינים:
  - תחום הכרות: כתלות בנראות (נראה נראויות שונות בהמשך הקורס), מוכרים בכל הקוד, ולא רק בתוך פונקציה מסויימת
  - משך קיום: אותו עותק של משתנה נוצר בזמן טעינת הקוד לזיכרון ונשאר קיים בזיכרון התוכנית כל עוד המחלקה בשימוש
  - אתחול: משתנים סטטיים מאותחלים בעת יצירתם. אם המתכנתת לא הגדירה להם ערך אתחול - יאותחלו לערך ברירת המחדל לפי טיפוסם (`0, false, null`)
  - הקצאת זיכרון: הזיכרון המוקצה להם נמצא באזור ה Heap (ולא באזור ה- Stack)

דוגמה: שימוש במשתנה גלובלי **counter** כדי לספור את מספר הקריאות למתודה **m()**

```
public class StaticMemberExample {
  public static int counter; //initialized by default to 0;

  public static void m() {
      int local = 0;
      counter++;
      local++;
      System.out.println("m(): local is " + local +
                          "\tcounter is " + counter);
  }

  public static void main(String[] args) {
      m();
      m();
      m();
      System.out.println("main(): m() was called " +
                                   counter + " times");
  }
}
```

שם מלא

- ניתן לפנות למשתנה counter גם מתוך קוד במחלקה אחרת, אולם יש צורך לציין את שמו המלא (qualified name)
- במחלקה שבה הוגדר משתנה גלובלי ניתן לגשת אליו תוך ציון שמו המלא או שם המזהה בלבד (unqualified name)
- בדומה לצורת הקריאה לשרותי מחלקה

```
public class AnotherClass {

    public static void main(String[] args) {
        StaticMemberExample.m();
        StaticMemberExample.m();
        StaticMemberExample.m();
        System.out.println("main(): m() was called " +
          StaticMemberExample.counter + " times");
        }
}
```

זה סופי

- ניתן לקבע ערך של משתנה (משתנה מקומי או משתנה מחלקה) ע"י ציון המשתנה כ **final**
- למשתנה שהוא **final** ניתן לבצע השמה פעם אחת בדיוק. כל השמה נוספת לאותו משתנה תגרור שגיאת קומפילציה
- דוגמה:

```
public final static long uniqueID = ++counter;
```

- מוסכמה מקובלת היא להגדיר שמות משתנים המציינים קבועים ב- UPPERCASE, כגון:

```
public final static double FOOT = 0.3048;
public final static double PI = 3.1415926535897932384;
```

התוכנית להיום

- תכנות בסיסי ב Java - בעבודה עצמית 2:
if/else, לולאות (for, while, do-while), לולאות על מערכים ולולאת for each
- בלוק switch case
- שירותי מחלקה, משתני מחלקה
- העמסה
- מודל הזיכרון של Java:
Heap and Stack, העברת ארגומנטים

העמסת שרותים (method overloading)

- לשתי פונקציות ב Java יכול להיות אותו שם (מזהה) גם אם הן באותה מחלקה, ובתנאי שהחתימה שלהן שונה
  - כלומר הן שונות בטיפוס ו\או במספר הארגומנטים שלהם
  - לא כולל ערך מוחזר!

- הגדרת שתי פונקציות באותו שם ובאותה מחלקה נקראת **העמסה**

- כבר השתמשנו בפונקציה מועמסת – `println` עבדה גם כשהעברנו לה משתנה פרימיטיבי וגם כשהעברנו לה מחרוזת

- נציג שלוש סיבות לשימוש בתכונת ההעמסה
  - נוחות
  - ערכי ברירת מחדל לארגומנטים
  - תאימות אחורה

העמסת פונקציות (שיקולי נוחות)

נממש את max המקבלת שני ארגומנטים ומחזירה את הגדול מביניהם

- ללא שימוש בתכונת ההעמסה (בשפת C למשל) היה צורך להמציא שם נפרד עבור כל אחת מהמתודות:

```
public class MyUtils {
  public static double max_double(double x, double y){ ...
  }

  public static long max_long(long x, long y){
     ...
  }
}
```

- השמות מלאכותיים ולא נוחים

העמסת פונקציות (פחות נוחות)

- בעזרת מנגנון ההעמסה ניתן להגדיר:

```
public static double max(double x, double y)
public static long max(long x, long y)
```

- בחלק מהמקרים, אנו משלמים על הנוחות הזו באי בהירות

- למשל, איזו מהפונקציות תופעל במקרים הבאים:

```
max(1L , 1L);         // max(long,long)
max(1.0 , 1.0);       // max(double,double)
max(1L , 1.0);        // max(double,double)
max(1 , 1);           // max(long,long)
```

העמסה והקומפיילר

- המהדר מנסה למצוא את הגרסה המתאימה ביותר עבור כל קריאה לפונקציה על פי טיפוסי הארגומנטים של הקריאה
- אם אין התאמה מדויקת לאף אחת מחתימות השרותים הקיימים, המהדר מנסה המרות (casting) ש(כמעט)-אינן מאבדות מידע.
- ראו פרק 5, **Conversions and Contexts** בקישור:

https://docs.oracle.com/javase/specs/jls/se21/html/jls-5.html

- אם לא נמצאת התאמה או שנמצאות שתי התאמות "באותה רמת סבירות" או שפרמטרים שונים מתאימים לפונקציות שונות המהדר מודיע על אי בהירות (ambiguity)

העמסה וערכי ברירת מחדל לארגומנטים

- ב python ניתן להגדיר פונקציות עם פרמטרים עם ערכי ברירת מחדל. למשל:

```
def func(x, y=1):
        return x+y
```

לפונקציה func ניתן לקרוא בשתי צורות: עם פרמטר יחיד ועם שני פרמטרים.

- ב Java לא ניתן להגדיר ערכי ברירת מחדל. במקום זאת, נשתמש בהעמסה:

```
public static int func(int x){
        return func(x, 1);
}

public static int func(int x, int y){
        return x + y;
}
```

העמסה ותאימות לאחור

- נניח כי במערכת כלשהי כבר קיימת הפונקציה השימושית compute המבצעת חישוב כלשהו על הארגומנט x.

```
public static  int compute(int x)
```

- לאחר זמן מה, כאשר הקוד כבר בשימוש במערכת (גם מתוך מחלקות אחרות שלא אתם כתבתם), עלה הצורך לבצע חישוב זה גם בבסיסי ספירה אחרים (החישוב המקורי בוצע בבסיס עשרוני)

- בשלב זה **לא ניתן להחליף** את חתימת הפונקציה להיות:

```
public static  int compute(int x, int base)
```

מכיוון שקטעי הקוד המשתמשים בפונקציה יפסיקו להתקמפל

העמסה, תאימות לאחור ושכפול קוד

- על כן במקום להחליף את חתימת השרות הוסיף נוסיף פונקציה חדשה בגרסה מועמסת
  - משתמשי הפונקציה המקורית לא נפגעים
  - משתמשים חדשים יכולים לבחור האם לקרוא לפונקציה המקורית או לגרסה החדשה ע"י העברת מספר ארגומנטים מתאים

- בעיה – קיים דמיון רב בין המימושים של הגרסאות המועמסות השונות (גוף המתודות compute)
- דמיון רב מדי - שכפול קוד זה הינו בעייתי מאוד

שכפול קוד הוא הדבר הנורא ביותר בעולם (התוכנה)!

העמסה, שכפול קוד ועקביות

- חסרונות שכפול קוד:

קוד שמופיע פעמיים, יש לתחזק פעמיים – כל שינוי, שדרוג או תיקון עתידי יש לבצע בצורה עקבית בכל המקומות שבהם מופיע אותו קטע הקוד

- בדי לשמור על עקביות שתי הגרסאות של **compute** נממש את הגרסה הישנה בעזרת הגרסה החדשה:

```
public static int compute(int x, int base){
  // complex calcultion...
}

public static int compute(int x){
   return compute(x, 10);
}
```

העמסת מספר כלשהו של ארגומנטים

- נניח שברצוננו לכתוב פונקציה שמחזירה את ממוצע הארגומנטים שקיבלה:

```
public static double average(double x){
  return x;
}

public static double average(double x1, double x2){
  return (x1 + x2) / 2;
}

public static double average(double x1, double x2, double x3){
  return (x1 + x2 + x3) / 3;
}
```

- למימוש 2 חסרונות:
  - שכפול קוד
  - לא תומך בממוצע של 4 ארגומנטים

העמסת מספר כלשהו של ארגומנטים

- רעיון: הארגומנטים יועברו במערך

```
public static double average(double [] args) {
   double sum = 0.0;
   for (double d : args) {
      sum += d;
   }
   return sum / args.length;
}
```

- יתרון: שכפול הקוד נפתר

- חסרון: הכבדה על הלקוח - כדי לקרוא לפונקציה יש ליצור מערך

```
public static void main(String[] args) {
   double [] arr = {1.0, 2.0, 3.0};
   System.out.println("Averge is:" + average(arr));
```

ג'אווה באה לעזרה

- ב Java קיים תחביר להגדרת שרות עם מספר לא ידוע של ארגומנטים (vararg)
- תחביר מיוחד של שלוש נקודות (...) יוצר את המערך מאחורי הקלעים:

```
public static double average(double ... args) {
  double sum = 0.0;
  for (double d : args) {
      sum += d;
  }
  return sum / args.length;
}
```

- ניתן כעת להעביר לשרות מערך או ארגומנטים בודדים:

```
double [] arr = {1.0, 2.0, 3.0};
System.out.println("Averge is:" + average(arr));
System.out.println("Averge is:" + average(1.0, 2.0, 3.0));
```

התוכנית להיום

תכנות בסיסי ב Java - בעבודה עצמית 2:
if/else, לולאות (for, while, do-while), לולאות על מערכים ולולאת for each

בלוק switch case

שירותי מחלקה, משתני מחלקה

העמסה

מודל הזיכרון של Java:
Heap and Stack, העברת ארגומנטים

העברת ארגומנטים

- כאשר מתבצעת קריאה לשרות, ערכי הארגומנטים נקשרים לפרמטרים הפורמליים של השרות לפי הסדר, ומתבצעת השמה לפני ביצוע גוף השרות.

- בהעברת ערך לשרות הערך **מועתק** לפרמטר הפורמלי

- צורה זאת של העברת פרמטרים נקראת **call by value**

- כאשר הארגומנט המועבר הוא **הפנייה** (התייחסות, reference) העברת הפרמטר **מעתיקה את ההתייחסות.**
  - בשפות תכנות אחרות ניתן לבצע העברה של המצביע עצמו

ב Java גם reference מועבר by value

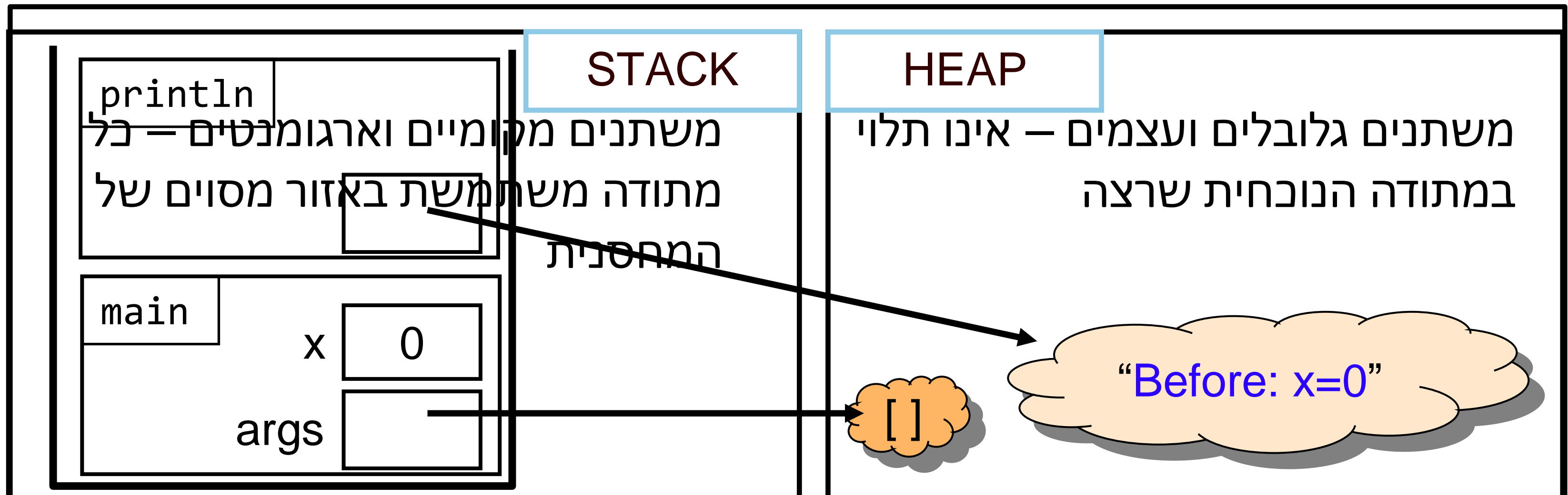
העברת פרמטרים by value

- העברת פרמטרים by value (ע"י העתקה) יוצרת מספר מקרים מבלבלים, שידרשו מאיתנו הכרות מעמיקה יותר עם מודל הזיכרון של Java
- למשל, מה מדפיס הקוד הבא?

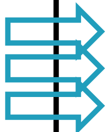
```
public class CallByValue {

   public static void setToFive(int arg){
      arg = 5;
   }

   public static void main(String[] args) {
      int x = 0;
      System.out.println("Before: x=" + x);
      setToFive(x);
      System.out.println("After: x=" + x);
   }
}
```

מודל הזיכרון של Java

Primitives by value

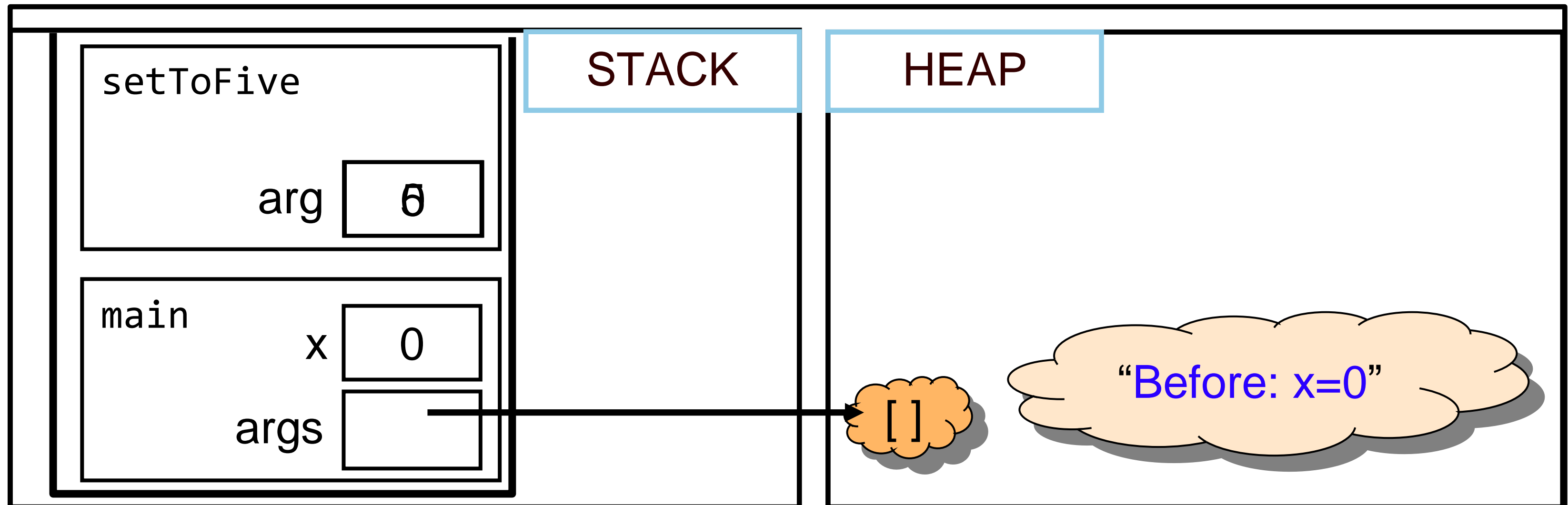


CODE



```
public class CallByValue {

    public static void setToFive(int arg){
        arg = 5;
    }

    public static void main(String[] args) {
        int x = 0;
        System.out.println("Before: x=" + x);
        setToFive(x);
        System.out.println("After: x=" + x);
    }
}
```

Primitives by value

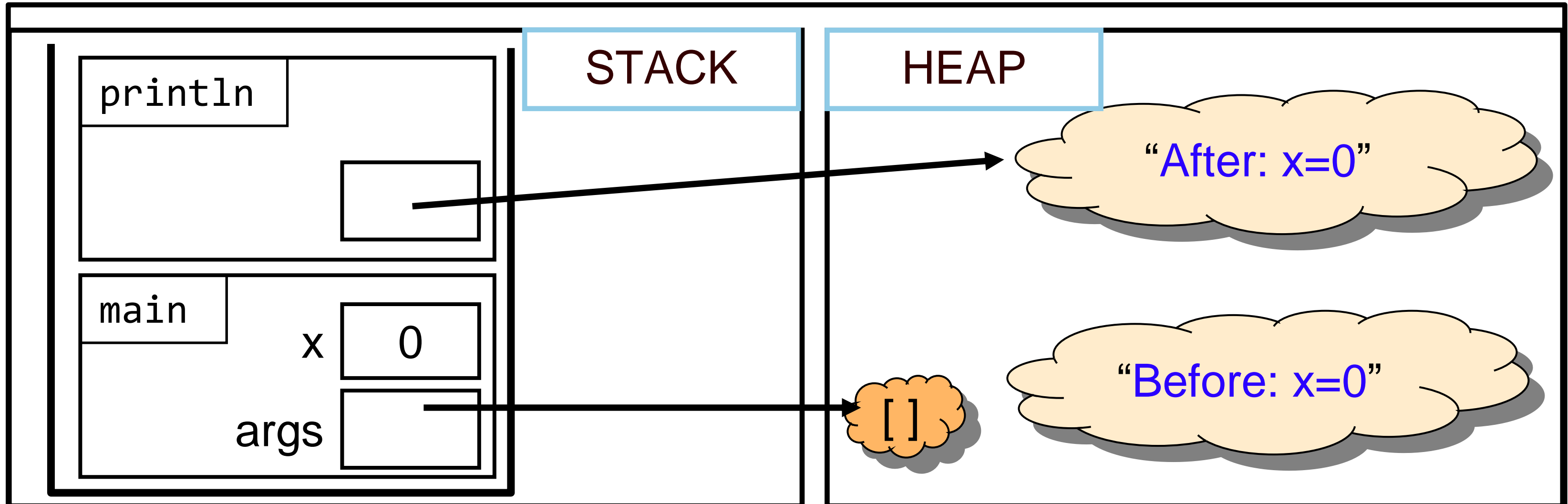


CODE

```java
public class CallByValue {

    public static void setToFive(int arg){
        arg = 5;
    }

    public static void main(String[] args) {
        int x = 0;
        System.out.println("Before: x=" + x);
        setToFive(x);
        System.out.println("After: x=" + x);
    }
}
```

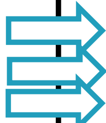
לאחר ש **setToFive** מסיימת את ריצתה המקום שהוקצה עבורה על ה-Stack משוחרר

Primitives by value



CODE

```
public class CallByValue {

    public static void setToFive(int arg){
        arg = 5;
    }

    public static void main(String[] args) {
        int x = 0;
        System.out.println("Before: x=" + x);
        setToFive(x);
        System.out.println("After: x=" + x);
    }
}
```



לאחר ש **main** מסיימת את
ריצתה המקום שהוקצה עבורה
על ה- Stack משוחרר

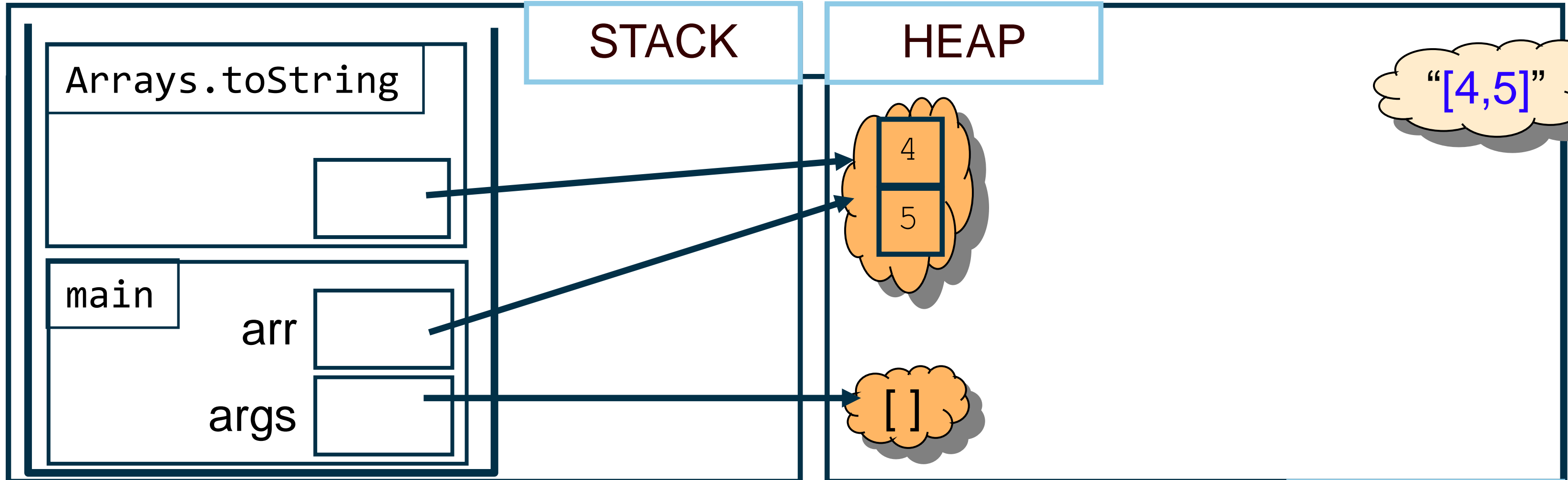
שמות מקומיים

- בדוגמא ראינו כי הפרמטר הפורמלי **`arg`** קיבל את הערך של הארגומנט x
- בחירת השמות השונים אינה משמעותית - יכולנו לקרוא לשני המשתנים באותו שם ולקבל התנהגות זהה
- שם של משתנה מקומי **מסתיר** משתנים בשם זהה הנמצאים בתחום עוטף או גלובליים
  - נראה את ההתנהגות הזו בשבוע הבא, בדוגמה של *בנאים*
- מתודה מכירה רק משתני מחסנית הנמצאים באזור שהוקצה לה על המחסנית (frame)

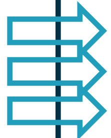
מה יקרה אם המשתנה המקומי שהועבר היה מטיפוס הפנייה? מה ידפיס הקוד הבא?

```
import java.util.Arrays; //explained later...

public class CallByValue {

   public static void setToZero(int [] arr){
      arr = new int[3];
   }

   public static void main(String[] args) {
      int [] arr = {4,5};
      System.out.println("Before: arr=" + Arrays.toString(arr));
      setToZero(arr);
      System.out.println("After: arr=" + Arrays.toString(arr));
   }
}
```

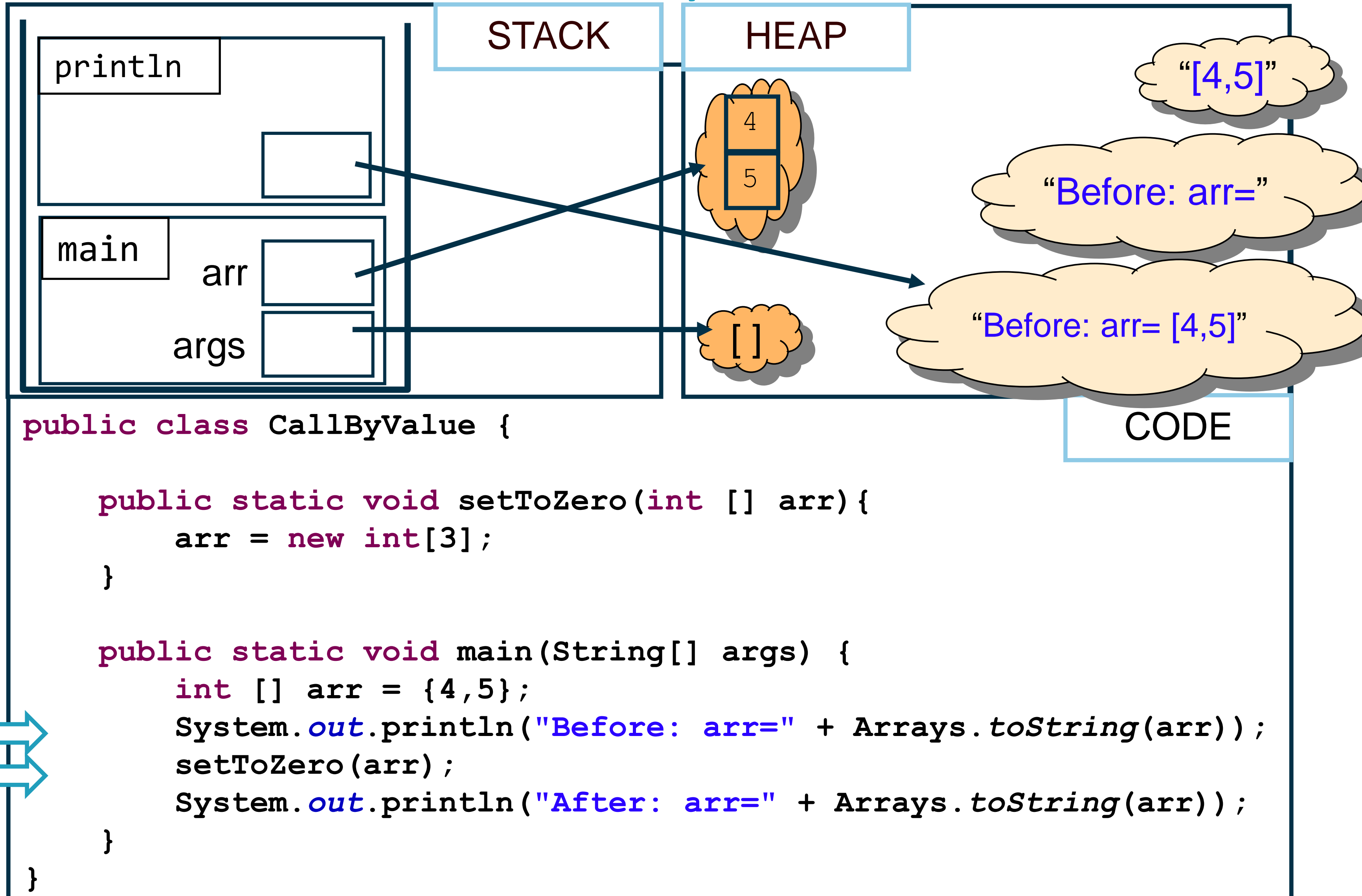
Reference by value




```
public class CallByValue {

    public static void setToZero(int [] arr){
        arr = new int[3];
    }

    public static void main(String[] args) {
        int [] arr = {4,5};
        System.out.println("Before: arr=" + Arrays.toString(arr));
        setToZero(arr);
        System.out.println("After: arr=" + Arrays.toString(arr));
    }
}
```

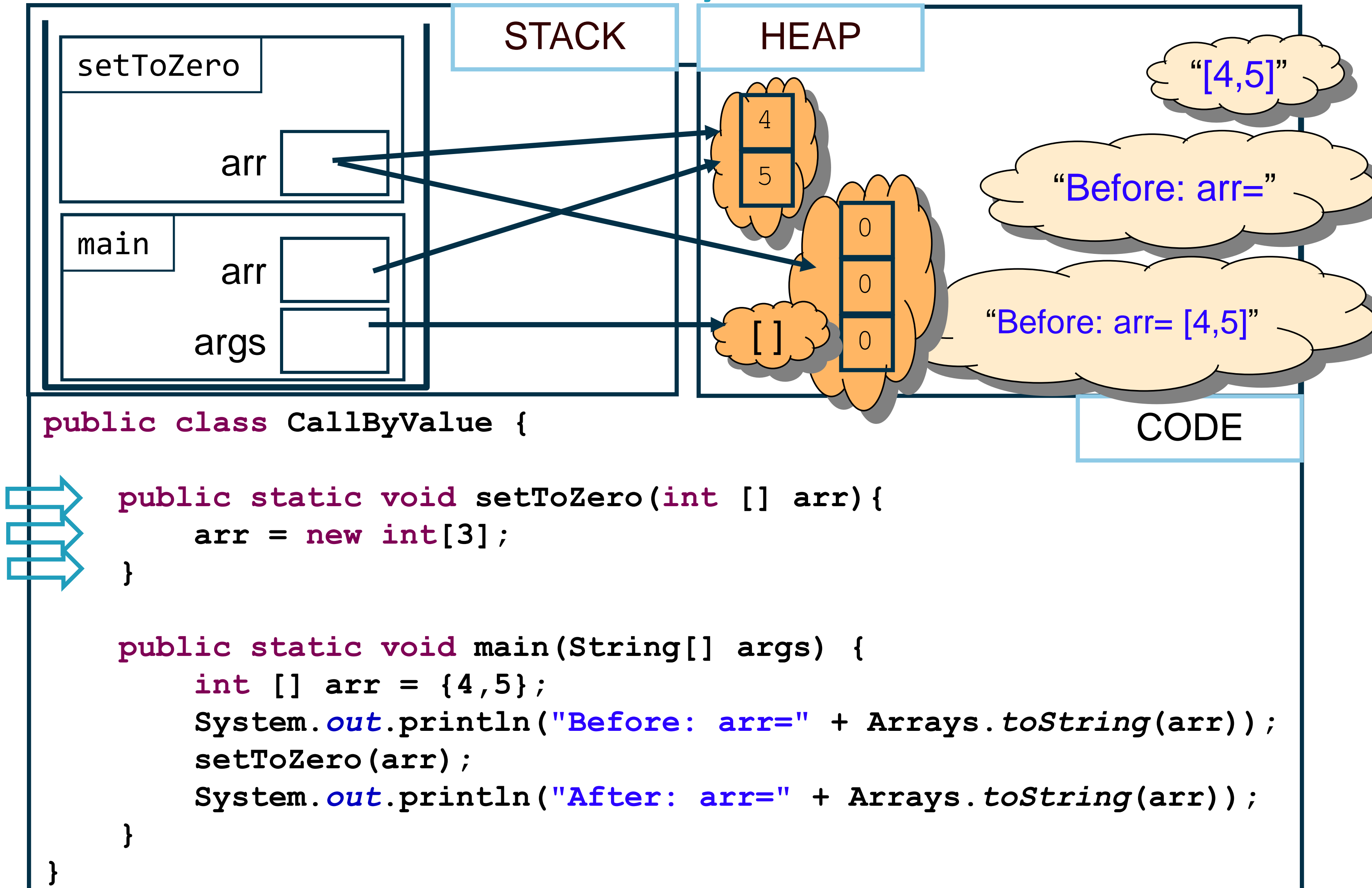


Reference by value

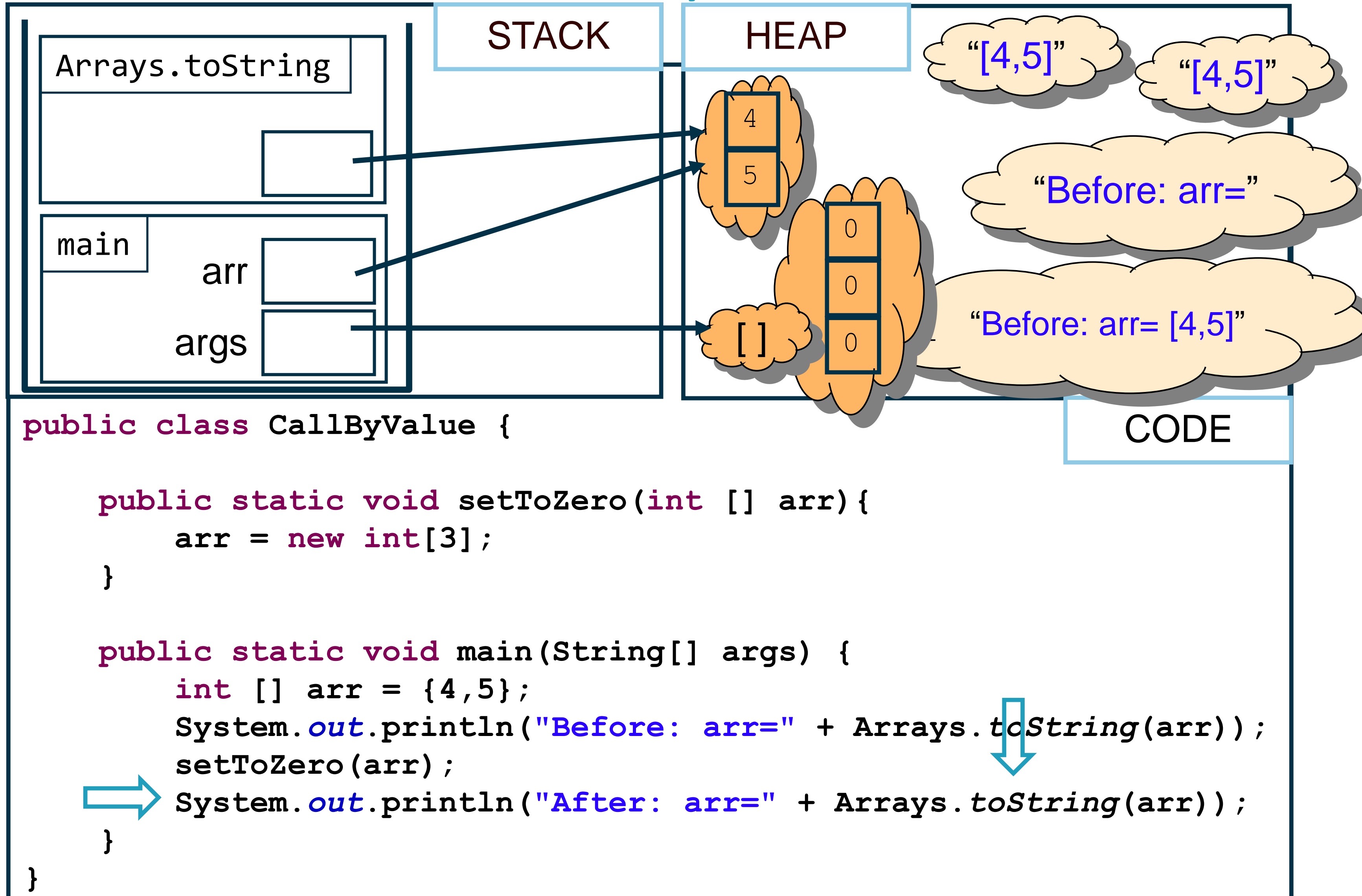



```
public class CallByValue {

    public static void setToZero(int [] arr){
        arr = new int[3];
    }

    public static void main(String[] args) {
        int [] arr = {4,5};
        System.out.println("Before: arr=" + Arrays.toString(arr));
        setToZero(arr);
        System.out.println("After: arr=" + Arrays.toString(arr));
    }
}
```

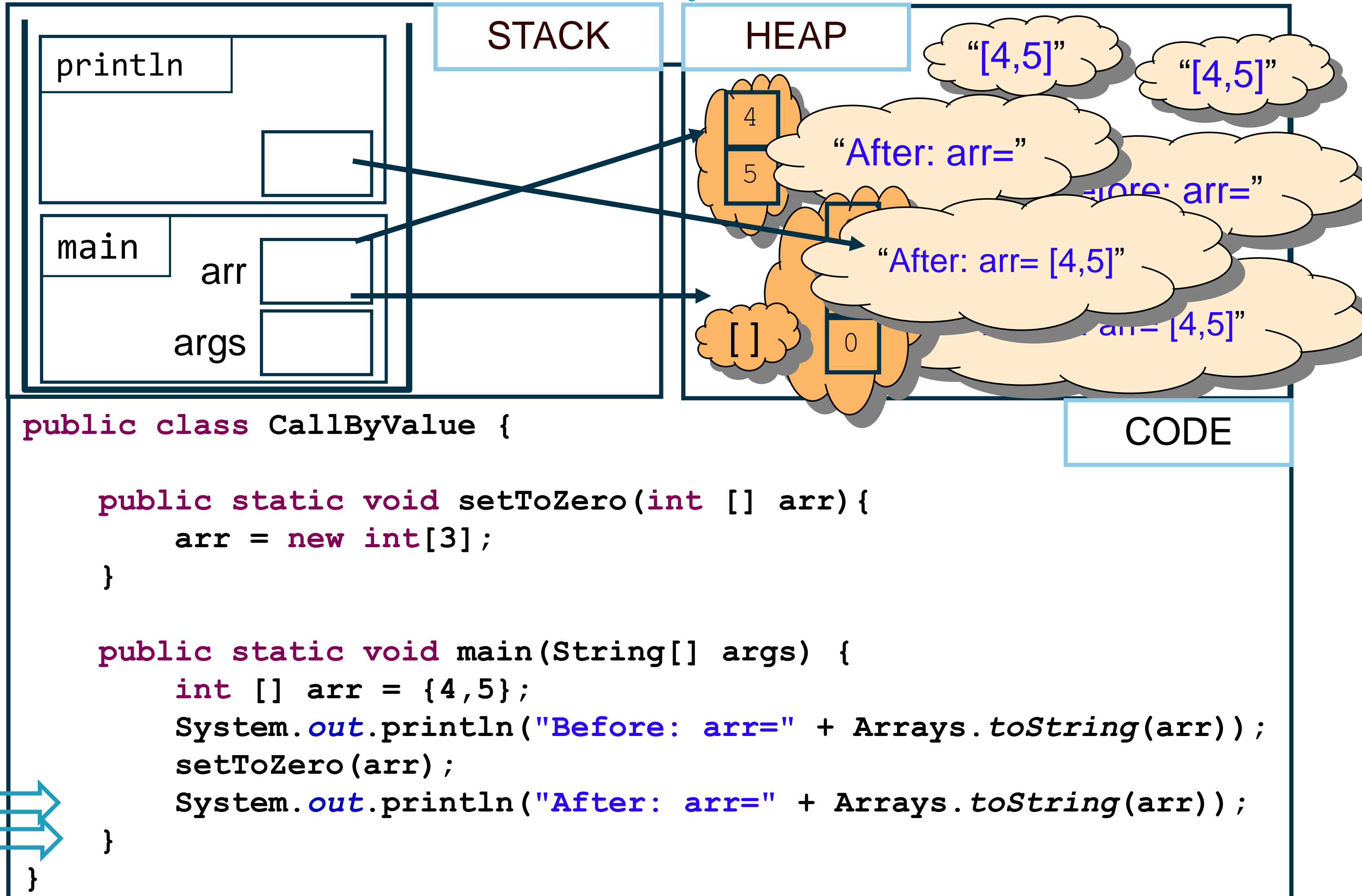
Reference by value



Reference by value



Reference by value



הפונקציה הנקראת והעולם שבחוץ

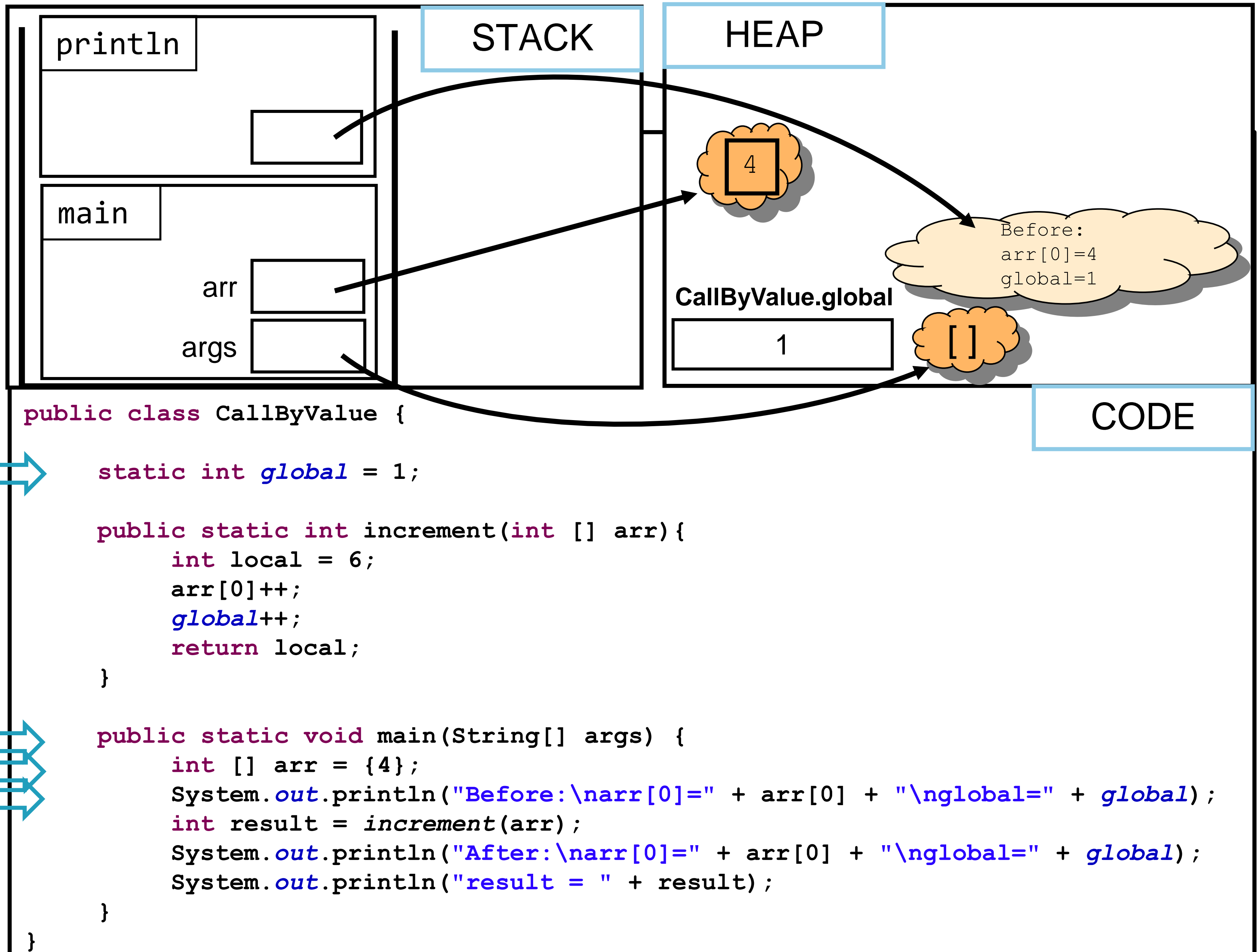
- בשיטת העברה by value לא יעזור למתודה לשנות את הארגומנט שקיבלה, מכיוון שהיא מקבלת עותק

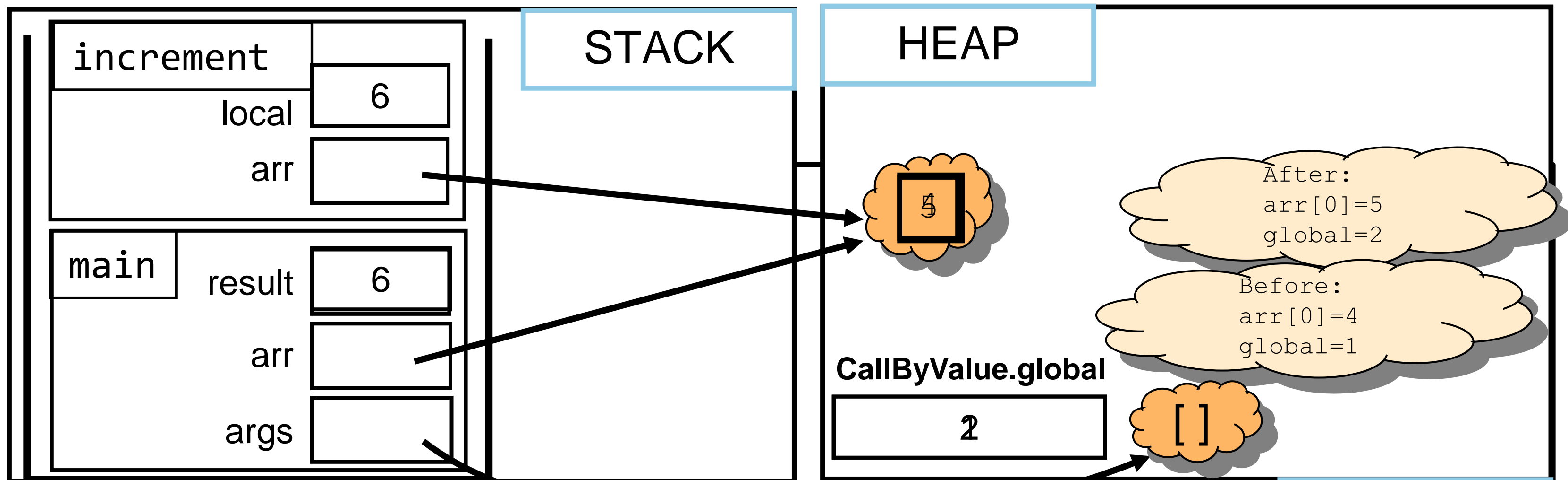
- אז איך יכולה מתודה להשפיע על ערכים במתודה שקראה לה?
  - ע"י ערך מוחזר
  - ע"י גישה למשתנים או עצמים שהוקצו ב- Heap

- מתודות שמשנות את תמונת הזיכרון נקראות בהקשרים מסוימים Mutators או Transformers

מה מדפיסה התוכנית הבאה?

```
public class CallByValue {

    static int global = 1;

    public static int increment(int [] arr){
        int local = 6;
        arr[0]++;
        global++;
        return local;
    }

    public static void main(String[] args) {
        int [] arr = {4};
        System.out.println("Before:\narr[0]=" + arr[0] +
                                "\nglobal=" + global);
        int result = increment(arr);
        System.out.println("After:\narr[0]=" + arr[0] +
                                "\nglobal=" + global);
        System.out.println("result = " + result);
    }
}
```


println
main
arr
args
STACK
HEAP
4
Before:
arr[0]=4
global=1
CallByValue.global
1
[ ]
CODE
public class CallByValue {
static int global = 1;
public static int increment(int [] arr){
int local = 6;
arr[0]++;
global++;
return local;
}
public static void main(String[] args) {
int [] arr = {4};
System.out.println("Before:\narr[0]=" + arr[0] + "\nglobal=" + global);
int result = increment(arr);
System.out.println("After:\narr[0]=" + arr[0] + "\nglobal=" + global);
System.out.println("result = " + result);
}
}




```
public class CallByValue {

    static int global = 1;

    public static int increment(int [] arr){
        int local = 6;
        arr[0]++;
        global++;
        return local;
    }

    public static void main(String[] args) {
        int [] arr = {4};
        System.out.println("Before:\narr[0]=" + arr[0] + "\nglobal=" + global);
        int result = increment(arr);
        System.out.println("After:\narr[0]=" + arr[0] + "\nglobal=" + global);
        System.out.println("result = " + result);
    }
}
```

Heap, Heap – Hooray!

STACK

HEAP

println

main

result 6

arr

args

result = 6

5

After:
arr[0]=5
global=2

Before:
arr[0]=4
global=1

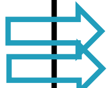
CallByValue.global

2

[ ]

CODE

```
public class CallByValue {

    static int global = 1;

    public static int increment(int [] arr){
        int local = 6;
        arr[0]++;
        global++;
        return local;
    }

    public static void main(String[] args) {
        int [] arr = {4};
        System.out.println("Before: arr[0]=" + arr[0] + "\tglobal=" + global);
        int result = increment(arr);
        System.out.println("After:  arr[0]=" + arr[0] + "\tglobal=" + global);
        System.out.println("result = " + result);
    }
}
```



משתני פלט (Output Parameters)

- איך נכתוב פונקציה שצריכה להחזיר יותר מערך אחד?
  - הפונקציה תחזיר מערך
- ומה אם הפונקציה צריכה להחזיר נתונים מטיפוסים שונים?
  - הפונקציה תקבל כארגומנטים הפניות לעצמים שהוקצו ע"י הקורא לפונקציה (למשל הפניות למערכים), ותמלא אותם בערכים משמעותיים
- ומה קורה אם נרצה שהפונקציה לא תחזיר ערך במקרים מסויימים?
  - החל מ Java 8 – המחלקה Optional עוזרת לדמות את ההתנהגות הרצויה, נראה אותה בהמשך הקורס.

גושי אתחול סטטיים

- ראינו כי אתחול המשתנה הסטטי התרחש מיד לאחר טעינת המחלקה לזיכרון, עוד לפני פונקציית ה main

- ניתן לבצע פעולות נוספות (בדרך כלל אתחולים למיניהם) מיד לאחר טעינת המחלקה לזיכרון, פעולות אלו יש לציין בתוך בלוק **`static`**
  - פרטים נוספים:
  https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html

תמונת הזיכרון האמיתית

- מודל הזיכרון שתואר כאן הוא פשטני – פרטים רבים נוספים נשמרים על המחסנית וב- Heap
- תמונת הזיכרון האמיתית והמדויקת היא תלוית סביבה ועשויה להשתנות בריצות בסביבות השונות
- נושא זה נידון בהרחבה בקורס קומפילציה