תוכנה 1 שיעור 4

מחלקות, עצמים, וחוזים

בשיעור הקודם: חוזים של שירותים

```
/**
 * finds the index of v in a if present
 *
 * @pre ???
 *
 * @post ???
 *
 *
 */

public static int findInSortedArray(int[] a, int v) {
   ...
}
```

בשיעור הקודם: חוזים של שירותים

```
/**
 * finds the index of v in a if present
 *
 * @pre a[i] < a[i+1]
 *
 * @post a[j] == v $implies $ret == j
 * @post a[j] != v for all j
 *                 $implies $ret == -1
 */

public static int findInSortedArray(int[] a, int v) {
  ...
}
```

עוד וריאציה (לחוזה)

```
/**
 * finds the index of v in a if present
 *
 * @pre a[i] <= a[i+1]
 *
 * @post $ret ==  j $implies a[j] == v
 * @post $ret == -1 $implies a[j] != v for all j
 * @post $ret >= -1 && $ret < a.length
 */

public static int findInSortedArray(int[] a, int v) {
  ...
}
```

תנאי קְדָם ואָחַר

- הלקוח (caller, client) אחראי לקיום תנאי הקדם
- לספק (callee, supplier) מותר להסתמך על תנאי הקדם
- אם תנאי הקדם מתקיים, הספק חייב לקיים את תנאי האחר
- אם תנאי הקדם לא מתקיים, הספק לא מחוייב לכלום (אבל מותר לו לקיים את תנאי האחר; שימושי רק במקרים קלים)
- הספק לא בודק את תנאי הקדם

היום

- מצב (state) ומשתמרים (invariants)
- הסתרה והכמסה
- פונקציית הפשטה
- משתמר מופשט ומשתמר ייצוג

חוזה למחלקה עם מצב

```
package tau.software1.lecture4;

public class CounterExample {
  public int c;

  public CounterExample() { c = 0; }

  public void increment() { c++; }

  /**
   * @pre none
   * @post $ret == #calls to increment()
   */
  public int counter() { return c; }

}
```

הוכחת נכונות?

הוכחת נכונות?

- נוכיח באינדוקציה כי הערך של c שווה למספר הקריאות ל-increment
- תכונה כזאת נקראת משתמר (invariant) כי היא אמורה להשתמר במשך כל חיי העצם (או התוכנית במקרים אחרים)
- אם זה מתקיים, אז תנאי האחר של counter יתקיים

ננסה להוכיח את המשתמר באינדוקציה

```
package tau.software1.lecture4;

public class CounterExample {
  public int c;

  public CounterExample() { c = 0; }

  public void increment() { c++; }

  /**
   * @pre none
   * @post $ret == #calls to increment()
   */
  public int counter() { return c; }

}
```

ננסה להוכיח את המשתמר באינדוקציה

- מקרה הבסיס: כאשר הבנאי חוזר, increment עדיין לא נקרא אף פעם, ואכן c==0
- נניח שהטענה נכונה אחרי k קריאות ל-increment
- אם קראו ל-increment k+1 פעמים, אז לפני הקריאה האחרונה התקיים c==k לפי הנחת האינדוקציה, ולכן בקריאה האחרונה הערך עלה ל-k+1
- הוכחנו

ההוכחה פגומה

- מקרה הבסיס: כאשר הבנאי חוזר, increment עדיין לא נקרא אף פעם, ואכן c==0
- נניח שהטענה נכונה אחרי k קריאות ל-increment
- אם קראו ל-increment k+1 פעמים, אז לפני הקריאה האחרונה התקיים c==k לפי הנחת האינדוקציה, ולכן בקריאה האחרונה הערך עלה ל-k+1
- הוכחנו

ההוכחה פגומה

- מקרה הבסיס: כאשר הבנאי חוזר, increment עדיין לא נקרא אף פעם, ואכן c==0
- נניח שהטענה נכונה אחרי k קריאות ל-increment
- אם קראו ל-increment k+1 פעמים, אז לפני הקריאה האחרונה התקיים c==k לפי הנחת האינדוקציה, ולכן בקריאה האחרונה הערך עלה ל-k+1
- לא הוכחנו; אולי במקום אחר בתוכנית שינו את c

עכשיו הקוד (וההוכחה) נכונים

השדה נגיש רק למחלקה הזאת

```
package tau.software1.lecture4;

public class Counter {
  private int c;

  public CounterExample() { c = 0; }

  public void increment() { c++; }

  /**
   * @pre none
   * @post $ret == #calls to increment()
   */
  public int counter() { return c; }

}
```

תיעוד יותר טוב של החוזה

```
public class Counter {
  /** @imp_inv c == #calls to increment */
  private int c;

  /** @pre none, @post none */
  public Counter() { c = 0; }

  /** @pre none, @post none */
  public void increment() { c++; }

  /** @pre none
   *  @post $ret == #calls to increment() */
  public int counter() { return c; }

}
```

משתמרים, בנאים, ושירותים

- הבנאי דואג שהמצב ההתחלתי של העצם יקיים את המשתמר של המחלקה
- המשתמש מתווסף בדרך כלל באופן סתום לתנאי הקדם של השירותים (לפחות הציבוריים, public)
- וגם לתנאי האחר שלהם!
- כלומר שירות יכול להניח את המשתמר אבל בתמורה חייב לדאוג שהוא יתקיים ביציאה מהשירות (לטובת הבאים אחריו)
- בזמן ריצה של שירות המשתמר לא תמיד מתקיים, רק בכניסה וביציאה

משתמר, הסתרה, הכמסה

- בעצמים שיש להם מצב (state, כלומר שדות) כדי לשמר את המשתמר הנראות (visibility) של השדות חייבת להיות פרטית; אי אפשר לקרוא או לשנות אותם ממחלקות אחרות
- זה גם מסתיר מלקוחות את הייצוג של המידע שהעצם מייצג (encasulation ,information hiding)
- אם רוצים שלקוחות יוכלו לקרוא את הייצוג (לשנות תמיד אסור אם יש משתמר לא טריויאלי), צריך להגדיר שירותי גישה ציבוריים (public) כמו counter()

שירותים פרטיים

- אפשר להגדיר גם את הניראות של שירותים כפרטית
- לקוחות (קוד במחלקות אחרות) לא יכולים לקרוא לשירותים כאלה
- כלומר אלה תמיד שירותי עזר של המחלקה עצמה
- לפעמים מגדירים אותם כדי למנוע שכפול קוד בתוך המחלקה או כדי שתהיה יותר קריאה
- מכיון שהם נקראים מתוך שירותים אחרים של המחלקה, לפעמים הם לא מניחים את המשתמר ולפעמים לא משחזרים אותו

נראות חבילה (אל תשתמשו)

```
package tau.software1.lecture4;

public class DontUsePackageVisibility {
  int c;

  public CounterExample() { c = 0; }

  public void increment() { c++; }

  /**
   * @pre none
   * @post $ret == #calls to increment()
   */
  public int counter() { return c; }

}
```

השדה נגיש רק למחלקות בחבילה הזאת

נראות חבילה (אל תשתמשו)

- ברירת המחדל של הנראות היא נראות לא שימושית שהיא באמצע בין ציבורית (public) ופרטית (private); מסומן על ידי <u>העדר</u> מילת מפתח לנראות
- כמעט אף פעם לא שימושי
  - מתירני מדי מכדי לאכוף את המשתמר (הוכחת נכונות צריכה לבדוק את כל החבילה)
  - ועדיין לא מאפשר גישה חופשית לשירות או לשדה

פונקציית הפשטה

מחסנית של מספרים שלמים

- תור LIFO (last in first out)
- פקודות push, pop, שאילתות pop, isEmpty

```
StackOfInts s1 = new StackOfInts();
System.out.println("isEmpty() == " + s1.isEmpty());

s1.push(1);
System.out.println("s1.top() == " + s1.top());

s1.push(2);
System.out.println("s1.top() == " + s1.top());

s1.pop();
System.out.println("s1.top() == " + s1.top());
System.out.println("isEmpty() == " + s1.isEmpty());
```

איך להגדיר את החוזה של המחלקה?

- אפשר לנסות להגדיר חוזה שיגדיר עבור הלקוח את הקשר בין השירותים השונים
- אבל זה קשה מאוד וסביר שהחוזה יהיה לא ברור
- למה?
- כי יש פיל בחדר ולא מדברים עליו
- הפיל הוא הרעיון המופשט של מחסנית שהעצם מייצג
- אם נגדיר פורמלית את הרעיון המופשט הזה יהיה קל להגדיר חוזה ברור

ייצוג פורמלי של מחסנית

- נייצג מחסנית כקבוצה סדורה (tuple) של שלמים, למשל (7, 33 , 45, 9, -3) כאשר האיבר הראשון בקבוצה (7) הוא האחרון שהוכנס והראשון שיצא
- פונקציית ההפשטה (abstraction function) של המחלקה ממפה עצם בזיכרון לייצוג המופשט

```java
/**  @abst (i_1, i_2, ... , i_n) or () for the empty stack */
public class StackOfInts {

    /** @abst AF(this) == () */
    public StackOfInts(){ ... }

    /**  @abst $ret == i_1  */
    public int top() { ... }

    /** @abst $ret == (AF(this) == ()) */
    public boolean isEmpty() { ... }

    /** @abst AF(this) == (i_2, i_3, ... , i_n) */
    public void pop() { ... }

    /** @abst AF(this) == (x, i_1, ... , i_n) */
    public void push(int x) { ... }

    /** @abst $ret == n */
    public int count() { ... }
}
```

```
/**  @abst (i1, i2, ... , in) or () for the empty stack */
public class StackOfInts {

    /** @abst AF(this) == () */
    public StackOfInts(){ ... }

    /**  @abst $ret == i1   */
    public int top() { ... }

    /** @abst $ret == (AF(this) == ()) */
    public boolean isEmpty() { ... }

    /** @abst AF(this) == (i2, i3, ... , in) */
    public void pop() { ... }

    /** @abst AF(this) == (x, i1, ... , in) */
    public void push(int x) { ... }

    /** @abst $ret == n */
    public int count() { ... }
}
```

סימון של המצב המופשט
לפני קריאה לשירות

```
/**  @abst (i1, i2, ... , in) or () for the empty stack */
public class StackOfInts {

    /** @abst AF(this) == () */
    public StackOfInts(){ ... }

    /**  @abst $ret == i1  */
    public int top() { ... }

    /** @abst $ret == (AF(this) == ()) */
    public boolean isEmpty() { ... }

    /** @abst AF(this) == (i2, i3, ... , in) */
    public void pop() { ... }

    /** @abst AF(this) == (x, i1, ... , in) */
    public void push(int x) { ... }

    /** @abst $ret == n */
    public int count() { ... }
}
```

עבור פקודות ובנאים,
מגדירים את המצב המופשט
אחרי הקריאה

```
/** @abst (i_1, i_2, ... , i_n) or () for the empty stack */
public class StackOfInts {

    /** @abst AF(this) == () */
    public StackOfInts(){ ... }

    /** @abst $ret == i_1 */
    public int top() { ... }

    /** @abst $ret == (AF(this) == ()) */
    public boolean isEmpty() { ... }

    /** @abst AF(this) == (i_2, i_3, ... , i_n) */
    public void pop() { ... }

    /** @abst AF(this) == (x, i_1, ... , i_n) */
    public void push(int x) { ... }

    /** @abst $ret == n */
    public int count() { ... }
}
```

עבור שאילתות, מגדירים איזה היבט של המצב המופשט מוחזר; הוא לא משתנה

יתכנו הרבה מימושים שונים

- יש הרבה דרכים לייצג את המצב המופשט ולממש את השירותים
- חלקן קלות למימוש, חלקן קשות
- חלקן יעילות (זמן חישוב, זיכרון, מקביליות) וחלקן פחות
- המימוש מגדיר את פונקציית ההפשטה: איך ממפים עצם בזיכרון למצב מופשט (מחסנית מופשטת)

מימוש אפשרי למחסנית

```
public class StackOfInts {

  public static int DEFAULT_STACK_CAPACITY = 10;
  private int[] rep;
  private int   t;

  public StackOfInts() {
    t   = -1;
    rep = new int[DEFAULT_STACK_CAPACITY];
  }

  public int top() { return rep[t]; }

  public boolean isEmpty() { return t == -1; }

  public void pop() { t--; }

  public int count() { return t + 1; }
  ...
```

מימוש אפשרי למחסנית

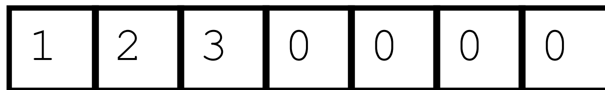
```
...
public void push(int x) {
    if (t == rep.length - 1) enlargeRep();
    t++;
    rep[t] = x;
}

/** allocate storage space in rep */
private void enlargeRep(){
    int[] biggerArr = new int[rep.length * 2];
    System.arraycopy(rep, 0, biggerArr, 0, rep.length);
    rep = biggerArr;
}

}
```

יש הרבה מימושים אחרים

- אנחנו בחרנו לייצג את המחסנית ע"י מערך שהאיבר הראשון בו הוא העמוק ביותר במחסנית



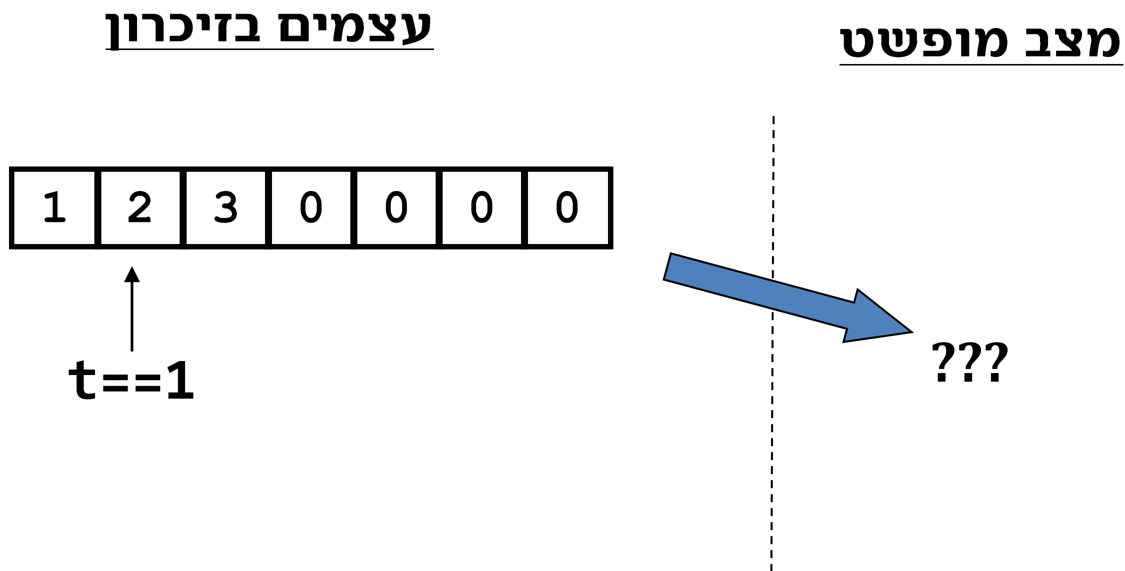
- אפשר להשתמש ברשימה מקושרת, אפשר במערך שגודלו בדיוק כגודל המחסנית, ועוד
- החוזה לא מספר ללקוח איך העצם המופשט מיוצג
- בדרך כלל ההסתרה הזאת מבורכת

יתרונות וחסרונות של הסתרת הייצוג

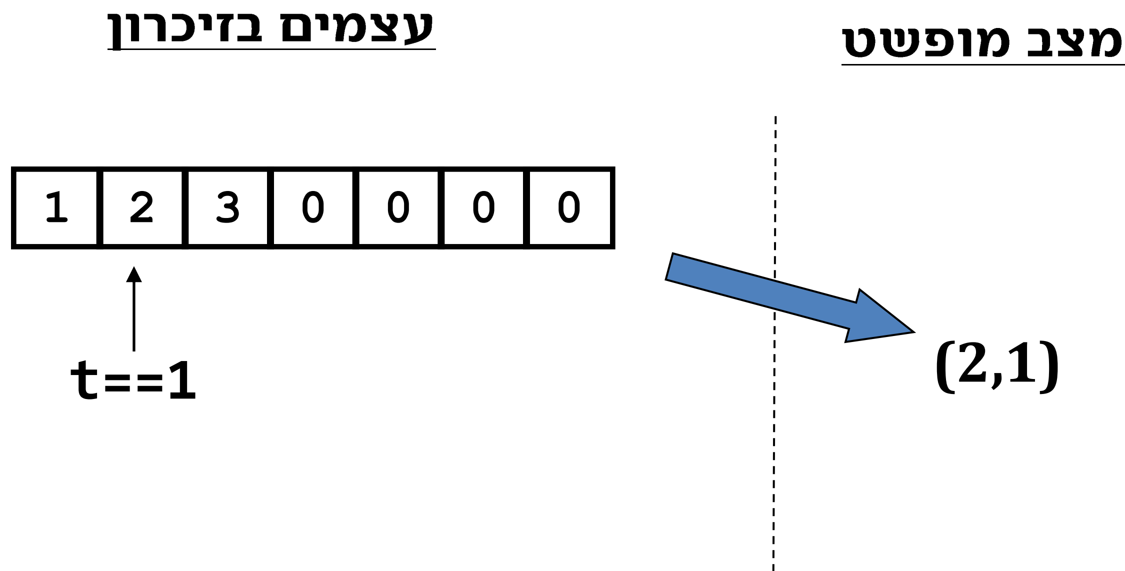
- היתרונות נובעים מזה שכאשר המימוש מוסתר, הלקוחות לא יכולים להניח עליו שום דבר, ולכן אפשר לשנות את הייצוג בלי להשפיע לרעה על לקוחות
- חוסר היכולת להניח הנחות על המימוש גורר גם חסרונות, בעיקר חוסר יכולת של לקוחות להסתמך על ביצועים (שימוש חסכוני בזיכרון וכו')
- למשל בספריה הסטנדרטית, TreeSet לעומת HashSet; פרטי המימוש עדיין מוסתרים והשדות פרטיים כדי שאפשר יהיה להוכיח נכונות ולשפר!

פונקציית ההפשטה של המימוש שלנו

פונקציית ההפשטה של המימוש שלנו



פונקציית ההפשטה של המימוש שלנו



פונקציית ההפשטה של המימוש שלנו

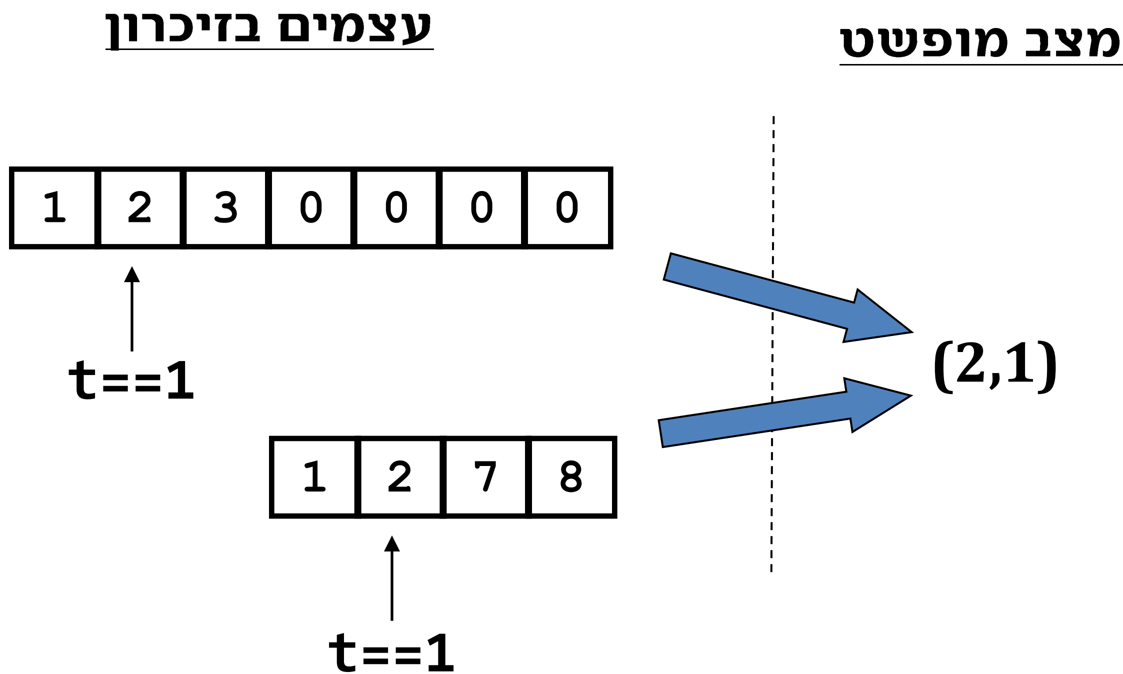
AF(`this`) $\equiv (x_1, \ldots, x_n)$

such that $\forall$ i in 1,…,n we have

x_i = `rep[t – i + 1]` and

n = `t + 1`

תמיד חד ערכית
(אבל לא תמיד חד-חד ערכית)

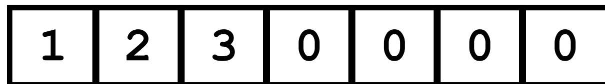


לא כל קונפיגורציה של שדות בזיכרון מתמפה למצב מופשט

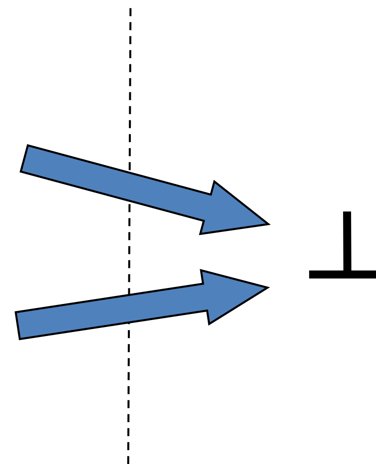
- הקונפיגורציות הללו של השדות אינן חוקיות; הן מפרות את **משתמר הייצוג** (rep invariant)

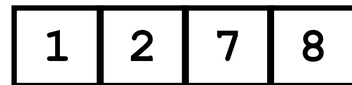
עצמים בזיכרון

מצב מופשט



t==35





t==-9

משתמר הייצוג של המחסנית

```
/** @imp_inv t < rep.length
 *  @imp_inv t >= -1
 *  @imp_inv isEmpty() || top() == rep[t]
 *  @imp_inv isEmpty() == (t==-1)
 */
public class StackOfInts {
```

- חלק מהטענות אפשר להחליף בתנאי בתר מימושי (implementation post condition), למשל

```
/** @imp_post $ret == rep[t] */
public int top() { ... }
```

איך מוכיחים נכונות של מחלקה (1)

שלב א: נוכיח כי כאשר נוצר עצם חדש, הוא מקיים את משתמר הייצוג

שלב ב: עבור כל שירות במחלקה נוכיח: אם מתקיים בכניסה לשירות תנאי הקדם וגם המשתמר מתקיים, אזי ביציאה מהשירות מתקיים תנאי האחר וגם המשתמר מתקיים

שלב ג: נוכיח כי פרט לשירותים של המחלקה, אין בתוכנית קוד שעשוי להפר את המשתמר אם הוא כבר מתקיים בדוגמא שלנו – אף אחד לא יכול לשנות את `rep` ו-`count` מחוץ למחלקה כי הנראות שלהם פרטית

איך מוכיחים נכונות (2): פונקציית הפשטה

- כאשר החוזה מוגדר במונחים של פונציית הפשטה, יש עוד שלב בהוכחה
- נניח שפקודה מסויימת m מתמירה מצב מופשט as למצב as', ושמצב מוחשי (בזיכרון) cs ממופה ל-as
- צריך להראות ש

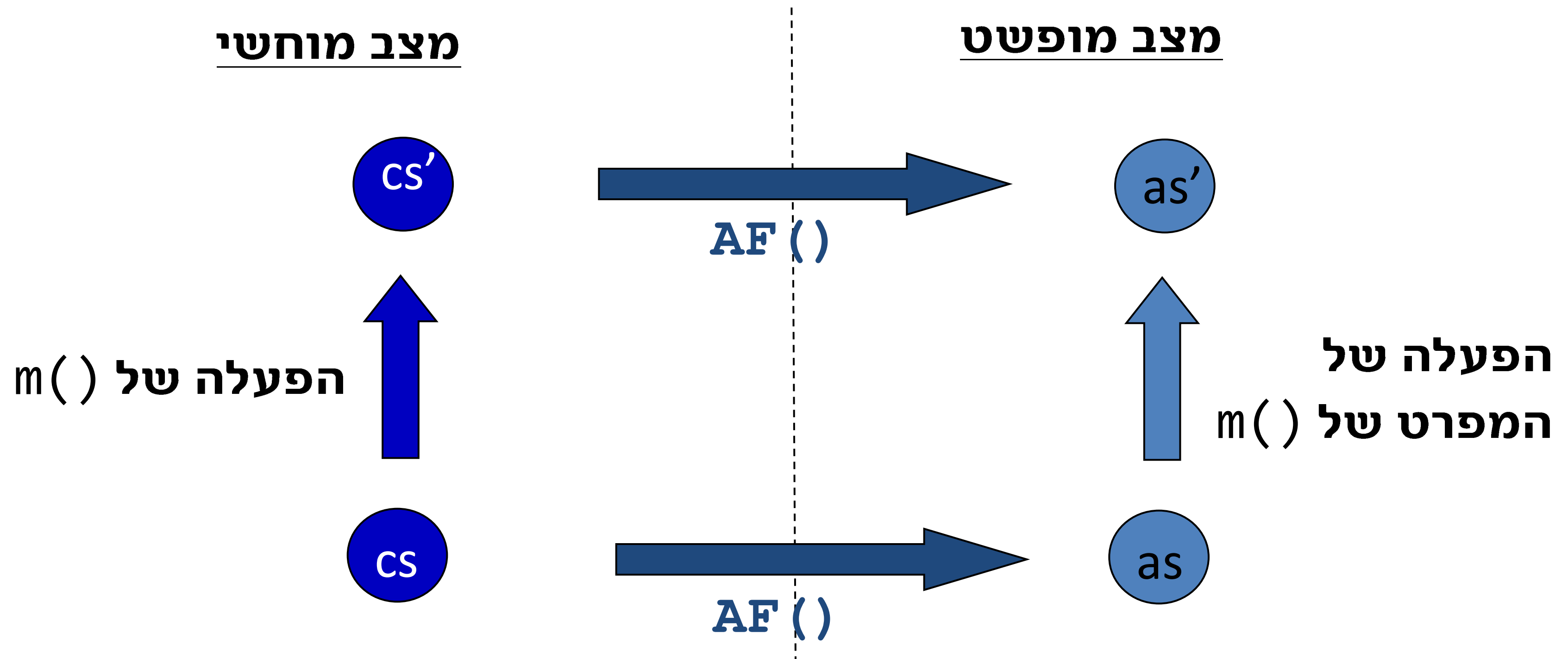
AF(cs.m()) == AF(cs).m()

- כי אז

AF(cs.m()) == AF(cs).m() == as.m() == as'

- וכנל לגבי כל המצבים האפשריים וכל הפקודות

כלומר המסלולים בתרשים תמיד שקולים



מתי אפשר (רצוי?) להימנע מהגדרת חוזה בעזרת פונקציית הפשטה

מתי אפשר (רצוי?) להימנע מהגדרת חוזה בעזרת פונקציית הפשטה

- כאשר יש שאילות שחושפות את כל המצב של העצם
- למשל, במחלקה שמייצגת נקודות במישור אפשר להגדיר

AF(this) = (this.getX(), this.getY())

- במקרה כזה אין צורך אמיתי בפונציית הפשטה
- השאילתות של מחסנית לא חושפות את כל המצב (המופשט) ולכן פונקציית ההפשטה עוזרת
- אפשר היה להגדיר פונקציות שחושפות את כל המצב (זה היה בעצם התפקיד של count), אבל זה היה מקלקל את העצם בכך שזה היה מונע מימושים יעילים מסויימים

מה פונקציית ההפשטה של

- מחלקה שמייצגת תאריכים
- שעון זמן אמיתי (שמתקדם גם כשלא קוראים לפקודה)
- זרוע רובוטית
- Map<String, Integer>
- ...

בנאים

- בונים ייצוג שמקיים את המשתמרים (ייצוג ומופשט)

```
public class StackOfInts {

  public static int DEFAULT_STACK_CAPACITY = 10;
  private int[] rep;
  private int   t;

  public StackOfInts() {
    t   = -1;
    rep = new int[DEFAULT_STACK_CAPACITY];
  }
```

אפשר להעמיס בנאים, אבל

- בונים ייצוג שמקיים את המשתמרים (ייצוג ומופשט)

```
public class StackOfInts {

  public static int DEFAULT_STACK_CAPACITY = 10;
  private int[] rep;
  private int   t;

  public StackOfInts() {
    t   = -1;
    rep = new int[DEFAULT_STACK_CAPACITY];
  }

  public StackOfInts(int expectedOccupancy) {
    t   = -1;
    rep = new int[expectedOccupancy];
  }
```

אפשר להעמיס בנאים, אבל

- שכפול הקוד עלול לגרום לבאגים

```
public class StackOfInts {

  public static int DEFAULT_STACK_CAPACITY = 10;
  private int[] rep;
  private int   t;

  public StackOfInts() {
    this( DEFAULT_STACK_CAPACITY );
  }

  public StackOfInts(int expectedOccupancy) {
    t   = -1;
    rep = new int[expectedOccupancy];
  }
```

אפשר להעמיס בנאים, אבל

- שכפול הקוד עלול לגרום לבאגים

```
public class StackOfInts {

  public static int DEFAULT_STACK_CAPACITY = 10;
  private int[] rep;
  private int   t;

  public StackOfInts() {
    this( DEFAULT_STACK_CAPACITY );
  }

  public StackOfInts(int expectedOccupancy) {
    t   = -1;
    rep = new int[expectedOccupancy];
  }
```

אם יש קריאה לבנאי אחר של המחלקה, חייבת להיות הפעולה הראשונה של הבנאי

מה היה לנו היום

- חזרה על תנאי קדם ואחר
- חוזים יותר מורכבים למחלקות שלמות
  - משתמר
  - שימור המשתמר בעזרת שיתוף פעולה של השירותים ובעזרת מניעת גישה ישירה לשדות הייצוג (נראות פרטית)
  - פונקציות הפשטה
  - הוכחת נכונות של מחלקה (עם או בלי פונקציית הפשטה)
  - בנאים והעמסה נכונה שלהם