SYNTAX
שפה מייצרת חשיבה

Classes & Objects

מחלקות ועצמים

SYNTAX
שפה מייצרת חשיבה

מה נלמד ?

1. פעולות בונות - Contructors
2. בקרת גישה והכמסה - Access level & encapsulation
3. העמסת פונקציות - Functions overloading
4. פעולות קובעות ומאחזרות - Getters & Setters
5. פעולות נוספות
6. יצירת מופעים של מחלקות - instances
7. מצביעים - References
8. משתנים ופעולות סטטיות - class and static variables
9. מחלקה בתוך מחלקה - Reference data members
10. מחרוזות ומערכים - String & Arrays

ששת השלבים ליצירת מחלקה

1. הכרזה על תכונת המחלקה
2. בניית הפעולה הבונה -constructor, לפעמים יהיה יותר מאחד.
3. בניית פונקציות ה- getters
4. בניית פונקציות ה- setters
5. בניית פונקציות שונות למחלקה
6. בניית פונקציית הדפסה

המחלקה Wallet

למחלקה 3 תכונות :

צבע

כמות כסף בארנק

כמות כסף שהארנק יכול להכיל

המחלקה Wallet

SYNTAX
שפה מייצרת חשיבה

constructor	public Wallet(int capacity, double currentMoney, String color)	הפעולה בונה ארנק עם צבע, תכולה וכמות כסף נוכחית.
getters	public int getCapacity()	מחזירה את ה-apacity של הארנק
	public double getCurrentMoney()	מחזירה את כמות הכסף בארנק
	public String getColor()	מחזירה את הצבע של הארנק
setter	public void setCapacity(int capacity)	הפעולה מקבלת עדכון על כמה כסף יכול הארנק להכיל ומעדכנת את תכונות האובייקט בהתאם
	public void setCurrentMoney (double currentMoney)	הפעולה מקבלת עדכון על כמה כסף יש בארנק ומעדכנת את הכסף בארנק בהתאם

שימו לב כי אין פעולה לעדכון צבע הארנק. מהרגע שנקבע בעת יצירתו אין למשתמש אפשרות לשנות אותו

בקרת גישה

- *public* - הגישה מותרת מכל מקום.
- *private* - הגישה מותרת רק מתוך המחלקה עצמה.
- *protected* - הגישה מותרת למחלקות היורשות/
- *package* - הגישה מותרת לכל מי ששייך לאותה

חבילה כמו המחלקה שלנו - **זוהי הבקרה האוטומטית**.

- הכמסהencapsulation

- תכונות האובייקט תמיד יהיו פרטיות - private
- כל גישה לתכונות באמצעות פונקציות מיוחדות ציבוריות - public
- הסתרת המימוש מהמשתמש. ניתן לעשות שינויים במימוש מבלי להשפיע על השימוש במחלקה.
- מאפשר לנו להסתיר חלק מהמשתנים ולתת גישה לאחרים.
- מאפשר לבצע בקרת קלט לפני ההשמה של ערך בתכונה.

המחלקה Wallet פעולות נוספות

Other functions

public void empty()	הפעולה מרוקנת את הארנק money=0
public boolean isEmpty()	הפעולה בודקת האם יש כסף בארנק ומחזירה אמת אחרת יוחזר שקר
public void fillMoney(double money)	הפעולה מקבלת סכום כסף ומוסיפה אותו לארנק
public void transferMoney(Wallet w)	הפעולה מקבלת ארנק ומעבירה אליו את כל הכסף שאפשר
public void print()	הפעולה מדפיסה תיאור של הארנק.

שלב ראשון - הכרזה על תכונות המחלקה

```
public class  Wallet
}
//    private class members
    private int capacity קיבולת הארנק// ;
    private double currentMoney כמות נוכחית// ;
    private String color צבע הארנק//;
```

הרשאת גישה פרטית (הכמסה)

טיפוס התכונה

שם התכונה

שלב 2 - הפעולה הבונה

SYNTAX
שפה מייצרת חשיבה

```
public class Wallet
}
        //private attributes
         private int capacity קיבולת הארנק//;
         private double currentMoney כמה כסף כרגע בארנק//;
         private String color צבע הארנק//;

    public Wallet(int capacity,double currentMoney,String color)
    }
                this.capacity=capacity;
                this.currentMoney=currentMoney;
                this.color=color;
    {
```

פעולה בונה
שם זהה לשם המחלקה. הפעולה נקראת שהעצם נבנה.הפעולה מאתחלת את ערכי התכונות.

כשם המחלקה

אתחול ראשוני של התכונות

מצביע מיוחד המתייחס this לעצמי המהדר יוצר אותו עבורנו

העמסת פונקציות - overloading

פונקציות בעלות אותו שם וכמות פרמטרים, טיפוסים וסדר שונה.

```
public Wallet(int capacity, double currentMoney, String color)}
    this.capacity=capacity;
        this.currentMoney=currentMoney;
        this.color=color;
{
public Wallet(int capacity)}
        this(capacity,0,"red");
{
public Wallet} ()
        this(100);
{
```

this משמשת גם לקריאה לפונקציות שלי ולא רק למשתנים שלי. הקריאה חייבת להיות השורה הראשונה בפעולה הבונה

copy constructor

פעולה בונה המקבלת עצם קיים כפרמטר ומעתיקה
את ערכי תכונותיו לתכונות העצם החדש.
לשים לב שישנה העתקה של ערכי התכונות.

```
public Wallet(Wallet other)
}
    this.capacity = other.capacity;
    this.currentMoney = other.currentMoney;
    this.color = other.color;
{
```

Empty constructor
ניתן להגדיר constructor ריק

```
public Wallet()
}
    this.capacity;0 =
    this.currentMoney = 0;
    this.color=null;
{
```

אם לא הגדרנו פעולה בונה בעצמנו המהדר מוסיף לנו לקוד פעולה בונה ריקה שרק מאתחלת את התכונות.

ברגע שנוסיף ולו פעולה בונה אחת בעצמנו אז המהדר לא יבנה עבורנו פעולה בונה ריקה, הוא יניח שלא שכחנו ולכן לא יוסיף עבורנו

syntax.org.il Copyright © 2016 Eran Katsav & Syntax College. Powered by

שלב 3 - getters

SYNTAX
שפה מייצרת חשיבה

```
public class Wallet
}
//    private attributes
    private int capacityקיבולת הארנק //;
    private double currentMoneyכמה כסף כרגע בארנק//;
    private String colorצבע הארנק//;

 פעולות נוספות...//.
public int getCapacity()
}
                return this.capacity;
{
public double getCurrentMoney()
}
        return this.currentMoney;
{
public String getColor()
}
        return this.color;
{
```

שלב 4 - setters

SYNTAX
שפה מייצרת חשיבה

```
public class Wallet
}
//    private attributes
    private in capacityקיבולת הארנק //;
    private double currentMoneyכמה כסף כרגע בארנק//;
    private String colorצבע הארנק//;
 פעולות נוספות...//.
public void setCapacity(int capacity)
}
                if(capacity>0) this.capacity=capacity;
{
public void setCurrentMoney (double currentMoney)
}
        if(currentMoney>=0 && currentMoney<=capacity)
                this.currentMoney=currentMoney;
{
public void  setColor(String color)
}
                this.color=color;
{
```

שלב 5 - פעולות נוספות

SYNTAX
שפה מייצרת חשיבה

```
public class Wallet
}
//    private attributes
    private in capacity קיבולת הארנק //;
    private double currentMoney כמה כסף כרגע בארנק//;
    private String color צבע הארנק//;
 //...פעולות נוספות
     public boolean isEmpty()
}
            return (this.currentMoney == 0);
{


{
```

שלב 5 - פעולות נוספות

SYNTAX
שפה מייצרת חשיבה

```
public class Wallet
}
//...תכונות
    public void fillMoney(double moneyToFill)
    }
      if (this.capacity < this.currentMoney + moneyToFill)
          this.currentMoney=this.capacity//ממלאים למקסימום ;
      else
              this.currentMoney += moneyToFill//מוסיפים ;
    {

{
```

פעולה המקבלת reference כפרמטר

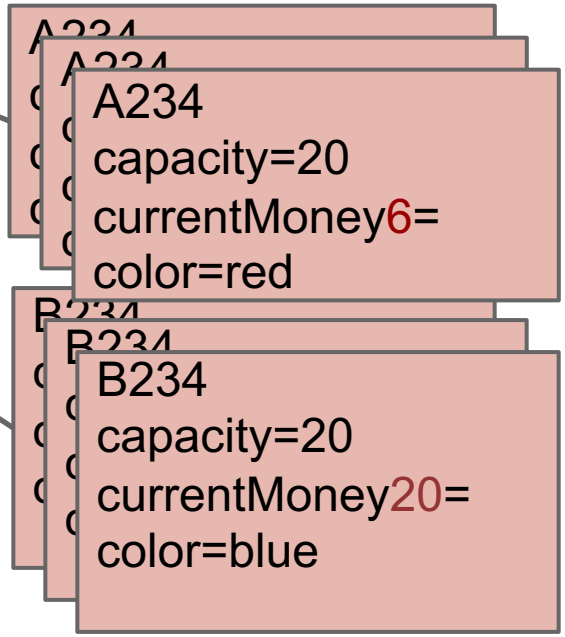
```
public void transferMoney (Wallet w)
}
    double freeSpace = w.getCapacity() – w.getCurrentMoney ;()
    if (this.currentMoney < freeSpace)
    {   //transfer all of our money
        w.fillMoney(this.currentMoney);
         this.currentMoney = 0;
    {
    else
    { // transfer only the availiable amount
        w.fillMoney(freeSpace);
        this.currentMoney – = freeSpace;
    {
{
```

יצירת עצמים בעזרת new

- כאשר נרצה לייצר עצם instance שהוא ישות של מחלקה נשתמש במלה השמורה (אופרטור) new.
- מה שיקרה הוא שמערכת ההפעלה "תלך" למקום בזיכרון שנקרא heap - זכרון ארוך טווח אך איטי יותר (לעומת ה stack שמכיל את המשתנים הפרימיטיביים) ותחפש מקום שיכול להכיל את האובייקט.
- כאשר היא תמצא כזה היא תייצר עצם, תקרא לפונקציה הבונה המתאימה ותחזיר לנו **כתובת reference** למקום הזה בזיכרון.

שימוש בפעולות

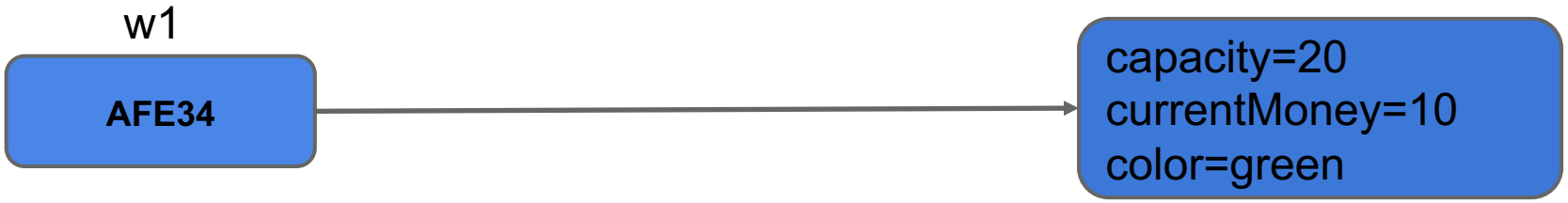
```
public static void main(String[] args)
}
    Wallet w1 = new Wallet(20,10,”red”) ;
    Wallet w2 = new Wallet(20,5,”blue”) ;
    w1.fill (6);
    w2.fill (5);
    w1.transferMoney(w2);

{
```



References

SYNTAX
שפה מייצרת חשיבה

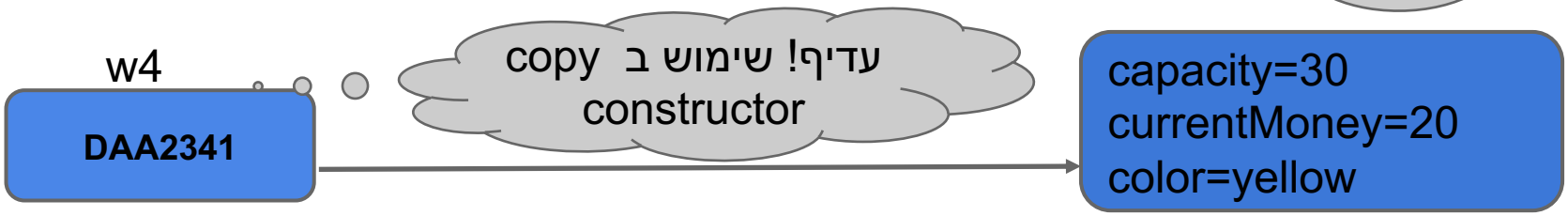
Wallet w1 = new Wallet(20,10,”green”);



Wallet w2 = new Wallet(30,20,”yellow”);

Wallet w3 = w2;

Wallet w4 = new Wallet(w2)



static סטטי -

1. static מלה השמורה לפני משתנים ופונקציות.
2. המשתנה או הפונקציה אינה שייכת לאובייקט ספציפי, אלא למחלקה עצמה.
3. כל האובייקטים יחלקו את אותם משתנים ופונקציות סטטיות (מקום אחד בזיכרון לכל האובייקטים).
4. פונים אליהם בעזרת ציון שם המחלקה ולא שם המשתנה.
5. פונקציות סטטיות יכולות לגשת אך ורק למשתנים סטטיים.

static example

```
class Wallet}
        //תכונות רגילות
        private static int counter// מונה למספר הארנקים שנוצרו ;

        public Wallet(...)
        }
                //אתחול התכונות
                counter++
    {
        public static int getCounter} ()
                return counter;
    {
{
```

הגדרת עצם ממחלקה ראשית

- פונקצית מחלקה רגילה נקראת בהקשר לעצם, ולכן מחייבת הגדרת עצם לפני הקריאה לה.

- הבעיה: היכן להגדיר עצם ממחלקה הראשית?
  בעיה זו מסוג "ביצה ותרנגולת".

  - כדי ליצור עצם (ביצה) יש לבצע פונקציה כלשהי במחלקה.
  - כדי לקרוא לפונקציה במחלקה יש צורך בעצם קיים.

הגדרת עצם ממחלקה ראשית

- הפתרון: פונקציה main של מחלקה ראשית היא סטטית, ולכן ניתן להגדיר בה עצמים מהמחלקה עצמה.

- פונקציה סטטית אינה נקראת בהקשר **לעצם**, אלא בהקשר **למחלקה**, ולכן אין צורך בעצם קיים כדי לקרוא לה.

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

- זו הסיבה שהפונקציה main הנקראת ע"י מערכת כנקודת התחלה של יישום חייבת להיות מוגדרת כסטטית: מפונקציה זו מתחילים להגדיר עצמים.

מחלקה בתוך מחלקהclass in class

1. חלק מהתכונות המחלקה יהיו מחלקה אחרת, כדוגמת מחלקת מכונית עם תכונה של רדיו.
2. לצורך כך נגדיר משתני מחלקה של מצביעים - references.
3. ניתן גם להגדיר מערך של אובייקטים כמשתנה מחלקה למשל המחלקה פלוגה אשר מכילה מערך של חיילים.
4. לשים לב שאתחול המערך אינו אומר שאתחלנו גם את התאים שלו.

class in class - example

נגדיר אובייקט מכונית ולאחר מכן אובייקט מוסך, לכל מוסך יש מקום חנייה ל25 מכוניות ולכן יכיל מערך של מכוניות.

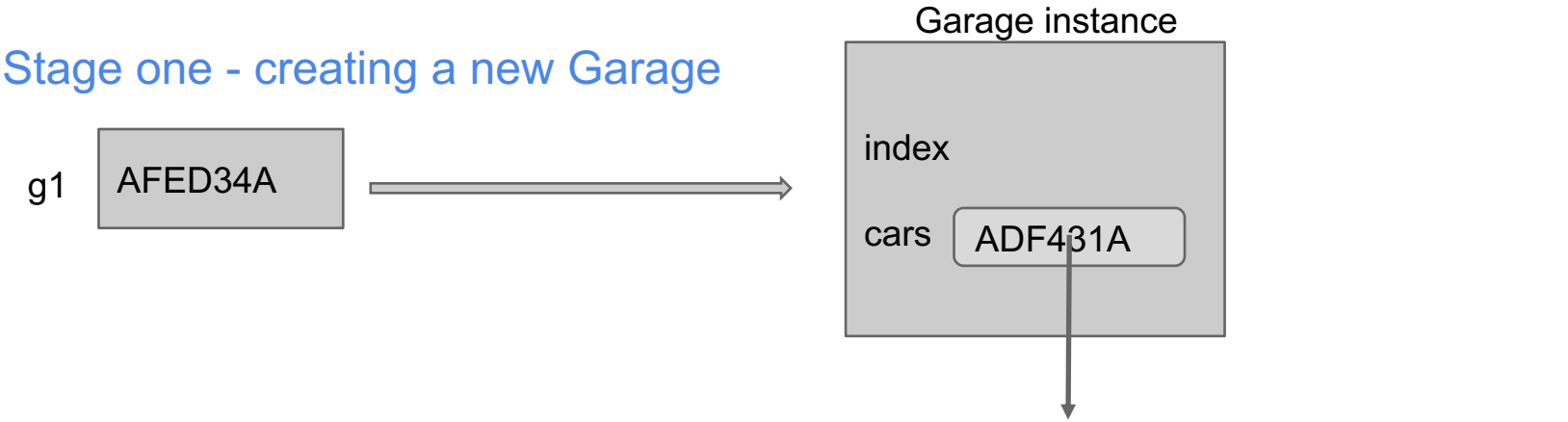
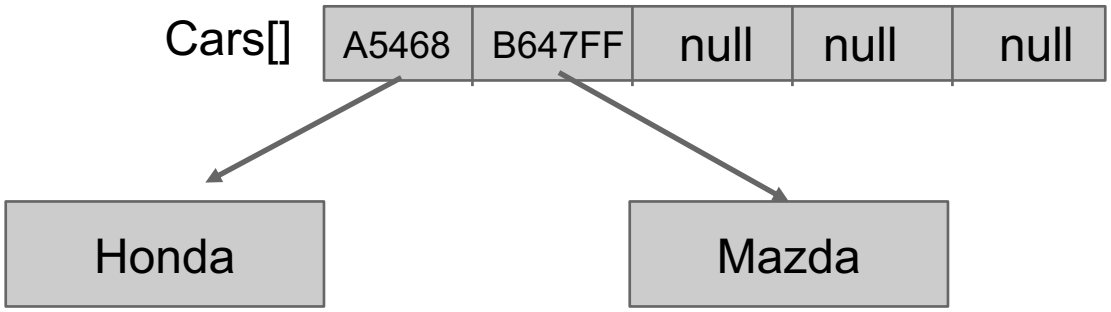
```
class Car{….}

class Garage}
        private Car[] cars;
        private int counter

        public Garage} ()
                cars  = new Car[25]
                counter=0;
    {
    public void addCar(Car c)}
        cars[counter]  = new Car(c);
        counter;++
    {
{
```

```
public static void main(String args[])
}
        //stage one
        Garage g1 = new Garage()

        //stage two
        g1.addCar(new Car("Honda"));
        g1.addCar(new Car("Mazda"));
{
```

SYNTAX
שפה מייצרת חשיבה

class in class - example - drawing

Stage one - creating a new Garage



Stage two - adding cars