

CSTA

הפנינג שונה

זה קורה רק פעם ב... הפנינג שונה של ביהמ"ד למדעי המחשב  
מפעל של סוף ביהמ"ד תלמידים ובוגרים באווירה משותפת!

**יום שישי 4.6.10 בשעה 15:00**

בגן הדיקרים (סמוך לבנין קולון)

60 אנשים. סוף ובוגרים של שפם ובניובנות  
בזמם אוזמנים אסוף שנה ולבוא אהנות

מאוזיקה טובה. ביהמ"ד קרה פירות פחיתות  
שונות וכמובן עם משהו לנשוס

**מחכים ומצפים לפגאנתכם!**

CSTA



# תוכנה 1 בשפת Java

equals, hashCode and wildcards

# תזכורת: המחלקה Object

```
package java.lang;

public class Object {
    public final native Class<?> getClass();

    public native int hashCode();

    public boolean equals(Object obj) {
        return (this == obj);
    }

    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return getClass().getName() + "@" +
            Integer.toHexString(hashCode());
    }
    ...
}
```

# מה יודפס?

```
public class Name {  
    ...  
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        List<Name> names = new ArrayList<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

false

false

- רצינו השוואה לפי תוכן אבל לא דרסנו את equals
- מימוש ברירת המחדל הוא השוואה של מצביעים

```
public class Object {  
    ...  
    public boolean equals(Object obj) {  
        return (this == obj);  
    }  
    ...  
}
```

# החזרה של equals

## רפלקסיבי

`x.equals(x)` יחזיר `true`

## סימטרי

`x.equals(y)` יחזיר `true` אם ורק אם `y.equals(x)` יחזיר `true`

## טרנזיטיבי

אם `x.equals(y)` ו-`y.equals(z)` יחזיר `true` גם `x.equals(z)` יחזיר `true`  
אז `x.equals(z)`

## עקבי

סדרת קריאות ל-`x.equals(y)` תחזיר `true` (או `false`) באופן עקבי  
אם מידע שדרוש לצורך ההשוואה לא השתנה

## השוואה ל null

`x.equals(null)` תמיד יחזיר `false`

# מתכון ל equals

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    Name other = (Name) obj;  
    return first.equals(other.first) &&  
        last.equals(other.last);  
}
```

1. ודאו כי הארגומנט אינו מצביע לאובייקט הנוכחי

2. ודאו כי הארגומנט אינו null

3. ודאו כי הארגומנט  
הוא מהטיפוס  
המתאים להשוואה

4. המירו את הארגומנט לטיפוס הנכון

5. לכל שדה "משמעותי", בידקו ששדה זה בארגומנט תואם לשדה באובייקט הנוכחי

# טעות נפוצה

■ להגדיר את הפונקציה equals כך:

```
public boolean equals(Name name) {  
    return first.equals(other.first) &&  
        last.equals(other.last);  
}
```

■ זו אינה דריסה (overriding) אלא העמסה  
(overloading)

■ שימוש ב @Override יפתור את הבעיה



# אז הכל בסדר?

```
public class Name {  
    ...  
    @Override public equals(Object obj) {  
        ...  
    }  
  
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        List<Name> names = new ArrayList<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

true יודפס

true יודפס

```
public class Name {  
    ...  
    @Override public equals(Object obj) {  
        ...  
    }  
  
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        Set<Name> names = new HashSet<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

true יודפס

false יודפס

# hashCode | equals

---

חובה לדרוס את hashCode בכל מחלקה  
שדורסת את equals!

# החזרה של hashCode

## עקביות

- מחזירה אותו ערך עבור כל הקריאות באותה ריצה, אלא אם השתנה מידע שבשימוש בהשוואת **equals** של המחלקה

## שוויון

- אם שני אובייקטים שווים לפי הגדרת equals אזי hashCode תחזיר ערך זהה עבורם

## חוסר שוויון

- אם שני אובייקטים אינם שווים לפי equals לא מובטח ש hashCode תחזיר ערכים שונים
- החזרת ערכים שונים יכולה לשפר ביצועים של מבני נתונים המבוססים על hashing (לדוגמא, HashSet ו HashMap)

# מימוש hashCode

```
@Override public int hashCode() {  
    return 31 * first.hashCode() + last.hashCode();  
}
```

■ השתדלו לייצר hash כך שלאובייקטים שונים יהיה ערך hash שונה

■ המימוש החוקי הגרוע ביותר (לעולם לא לממש כך!)

```
@Override public int hashCode() {  
    return 42;  
}
```

# תמיכה באקליפס

- אקליפס תומך ביצירה אוטומטית (ומשולבת) של hashCode ו equals
- בתפריט Source ניתן למצוא Generate hashCode() and equals()

# מערכים ורשימות

האם הקוד הבא מתקמפל? ■

```
String[] as = new String[10];  
Object[] ao = as;
```

וזה? ■

```
List<String> ls = new ArrayList<String>(10);  
List<Object> lo = ls;
```

# תזכורת

מערכים הם קו-וריאנטים

אם `Sub` הוא תת-טיפוס של `Super` אז `Sub[]` הוא תת-טיפוס של `Super[]`

✓ `Sub[] sub = ...`  
`Super[] sup = sub;`

טיפוסים גנריים הם וריאנטים

אם `T1` ו `T2` טיפוסים שונים אז, לדוגמה, בין הטיפוסים `List<T1>` ו `List<T2>` לא מתקיים יחס של תתי-טיפוסים גם אם יחס כזה מתקיים בין `T1` ו `T2`

✗ `List<T1> sub = new ArrayList<T1>();`  
`List<T2> sup = sub;`



# משימה

נממש פונקציה המדפיסה את כל האברים שבאוסף ■

```
static void printCollection(Collection<Object> c) {  
    for (Object o : c)  
        System.out.println(o);  
}
```

האם ניתן להעביר לפונקציה `List<String>` כפרמטר? ■

```
public static void main(String[] args) {  
    List<String> ls = new ArrayList<String>();  
    // populate list  
    X printCollection(ls);  
}
```

נרצה פונקציה שמקבל collection של "כל טיפוס" ■

# האמא של כל ה-Collections

■ טיפוס העל של כל האוספים הוא:

`Collection<?>` - collection of unknown

■ זהו אוסף שטיפוס האברים שבו מתאים להכל

■ הטיפוס של האברים נקרא *wildcard type* מסיבות ברורות

```
static void printCollection(Collection<?> c) {  
    for (Object o : c)  
        System.out.println(o);  
}
```

# שימושים

- כשלא יודעים או לא אכפת לנו מהו הטיפוס האמיתי
- לדוגמא, פונקציות הפועלות על מבנה של collection (shuffle, rotate, ...)

```
static int numberOfElementsInCommon(Set<?> s1, Set<?> s2)
{
    int result = 0;
    for (Object o : s1) {
        if (s2.contains(o))
            result++;
    }
    return result;
}
```

# מחסנית

נתונה המחלקה:

```
public class Stack<E> {  
    public Stack() {...}  
    public void push(E e) {...}  
    public E pop() {...}  
    public boolean isEmpty() {...}  
}
```

נרצה להוסיף

```
public void pushAll(Collection<E> src) {  
    for (E e : src)  
        push(e);  
}
```

מה הבעיה במימוש?

# Unbounded Wildcard

```
static void printCollection(Collection<?> c) {  
    for (Object o : c)  
        System.out.println(o);  
}
```

תמיד נוכל לקרוא איברים ולהתייחס אליהם כ- Object

```
Collection<?> ls = new ArrayList<String>();  
X c.add(new Object());
```

מכיוון שאיננו יודעים מה טיפוס האברים באוסף c לא ניתן להוסיף אלמנטים.  
כל אובייקט שנעביר כפרמטר ל-add חייב להיות מתת-טיפוס של האבר, אבל  
איננו יודעים מהו טיפוס האבר.  
החריג היחיד הוא null

# הבעיה

מה קורה עבור הקוד הבא: ■

זיכרו Integer יורש מ Number ■

```
Stack<Number> numberStack = new Stack<Number>();  
Collection<Integer> integers = ...  
numberStack.pushAll(integers);
```

הודעת שגיאה ■

The method `pushAll(Collection<Number>)` in the type `Stack<Number>` is not applicable for the arguments `(Collection<Integer>)`

ממה נובעת הודעת השגיאה? ■

# ? extends E

טיפוס הקלט ל `pushAll` ■

במקום “Collection of E” נרצה ■

“Collection of **some subtype of E**”

```
public class Stack<E> {  
    ...  
    public void pushAll(Collection<? extends E> src) {  
        for (E e : src)  
            push(e);  
    }  
}
```

חסם עליון על טיפוס הקלט ■

E הוא תת טיפוס של עצמו ■

# popAll

כעת נרצה להוסיף את popAll ■

```
public class Stack<E> {  
    ...  
    public void popAll(Collection<E> dst) {  
        while (!isEmpty())  
            dst.add(pop());  
    }  
}
```

בעיית קומפילציה? ■

מה עם קוד הלקוח? ■



# קוד הלקוח

האם יש בעיה בקוד הלקוח? ■

✓ `Stack<Number> numberStack = new Stack<Number>();`  
`Object o = numberStack.pop();`

✗ `Collection<Object> objects = ...`  
`numberStack.popAll(objects);`

האם השימוש ב `extend` מתאים גם פה? ■

# ? super E

טיפוס הקלט ל popAll ■

במקום "Collection of E" נרצה ■

"Collection of **some supertype of E**"

```
public class Stack<E> {  
    ...  
    public void popAll(Collection<? super E> dst) {  
        while (!isEmpty())  
            dst.add(pop());  
    }  
}
```

חסם תחתון על טיפוס הקלט ■

E הוא תת טיפוס של עצמו ■

# get-put principal\*

■ השתמשו ב **extends** כאשר אתם קוראים נתונים ממבנה, ב **super** כאשר אתם מכניסים נתונים למבנה ואל תשתמשו ב wildcards כאשר אתם עושים את שניהם

■ ב pushAll קוראים נתונים מהמשתנה src

■ ב popAll מכניסים נתונים למשתנה dst

---

\* “*Java Generics and Collections*” by Naftalin and Wadler

# סיכום Wildcards

■ שלושה סוגים של wildcards:

1. ?

קבוצת "כל הטיפוסים" או "טיפוס לא ידוע כלשהו"

2. **T extends ?**

משפחת תתי הטיפוס של T (כולל T)

3. **T super ?**

משפחת טיפוס העל של T (כולל T)