

# תוכנה 1

תרגול 13 – סיכום

# בחינה באופק!

■ הבחינה תכלול את כל הנושאים שכיסינו במהלך

הסמסטר:

■ כל ההרצאות

■ כל תרגולים

■ כל תרגילי בית

■ חומר סגור

■ שאלות אמריקאיות

# קצת על מנשקים

- מנשק יכול להרחיב יותר ממנשק אחד
- שירותים במנשק הם תמיד ציבוריים, וכברירת מחדל מופשטים

```
public interface MyInterface {  
    public abstract int foo1(int i);  
    int foo2(int i);  
}
```

The modifiers of foo1 and foo2 are the same.

# מנשקים

```
public interface Foo {  
    public void bar() throws Exception;  
}
```

```
public class FooImpl implements Foo {  
    public void bar() {  
        System.out.println("No exception is thrown");  
    }  
}
```

```
public static void main(String args[]) {  
    Foo foo = new FooImpl();  
    foo.bar();  
}
```

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

שגיאת קומפילציה:  
"Unhandled exception type Exception"

# מנשקים - המשך

```
public interface Foo {  
    public void bar() throws Exception;  
}
```

```
public class FooImpl implements Foo {  
    public void bar() {  
        System.out.println("No exception is thrown");  
    }  
}
```

```
public static void main(String args[]) {  
    FooImpl foo = new FooImpl();  
    foo.bar();  
}
```

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

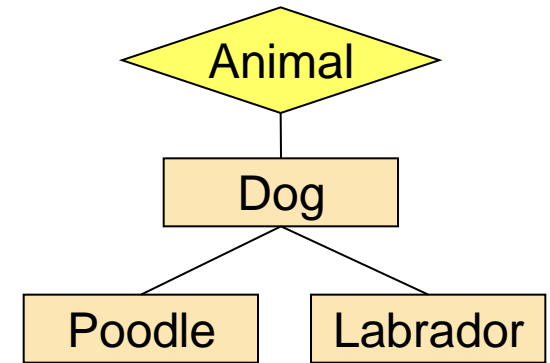
פלט:

"No exception is thrown"

# מנשקים וירושה

Consider the following class hierarchy:

```
Interface Animal {...}
class Dog implements Animal {...}
class Poodle extends Dog {...}
class Labrador extends Dog {...}
```



Which of the following lines (if any) will not compile?

```
Poodle poodle = new Poodle();
Animal animal = (Animal) poodle;
Dog dog = new Labrador();
animal = dog;
poodle = dog;
```

poodle = (Poodle) dog;  
-No compilation error  
-Runtime Exception

- Compilation Error  
Type mismatch: cannot convert

Labrador labrador = (Labrador) dog;  
-No compilation error  
-No Runtime Exception



# מנשקים וירחשה

```
class A {  
    public void print() {  
        System.out.println("A");  
    }  
}
```

האם יש שגיאה?

```
class B extends A implements C {  
}
```

```
interface C {  
    void print();  
}
```

אין שגיאות קומפילציה

public כברירת מחדל

# מנשקים וירושה



```
class A {  
    void print() {  
        System.out.println("A");  
    }  
}
```

האם יש שגיאה?

```
class B extends A implements C {  
}
```

```
interface C {  
    void print();  
}
```

שגיאת קומפילציה:  
The inherited package method A.print() cannot hide the public abstract method in C



The following table shows the access to members permitted by each modifier

Access Levels

Modifier	Class	Package	Subclass	World
<code>public</code>	Y	Y	Y	Y
<code>protected</code>	Y	Y	Y	N
<i>no modifier</i>	Y	Y	N	N
<code>private</code>	Y	N	N	N

# דריסה של שירותים

```
public class A {  
    public void print() {  
        System.out.println("A");  
    }  
}
```

```
public class B extends A {  
    public void print() {  
        System.out.println("B");  
    }  
}
```

הפלט:

B  
B

```
public class C {  
    public static void main(String[]  
        args){  
        B b = new B();  
        A a = b;  
  
        b.print();  
        a.print();  
    }  
}
```

אין צורך ב-  
casting

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

# דריסה של שירותים וניראות

```
public class A {  
    public void print() {  
        System.out.println("A");  
    }  
}
```

```
public class B extends A {  
    protected void print() {  
        System.out.println("B");  
    }  
}
```

שגיאת קומפילציה:  
"Cannot reduce the  
visibility of the  
inherited method from A"

```
public class C {  
    public static void main(String[]  
        args) {  
        B b = new B();  
        b.print();  
    }  
}
```

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

## דריסה של שירותים וניראות (2)

```
public class A {  
    protected void print() {  
        System.out.println("A");  
    }  
}
```

```
public class B extends A {  
    public void print() {  
        System.out.println("B");  
    }  
}
```

```
public class C {  
    public static void main(String[]  
        args) {  
        B b = new B();  
        b.print();  
    }  
}
```

הפלט:  
B

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

# הורשה

```
public class A {  
    public void foo() {  
        System.out.println("A.foo()");  
    }  
  
    public void bar() {  
        System.out.println("A.bar()");  
        foo();  
    }  
}
```

```
public class B extends A {  
    public void foo() {  
        System.out.println("B.foo()");  
    }  
  
    public static void main(String[]  
        args) {  
        A a = new B();  
        a.bar();  
    }  
}
```

הפלט:

```
A.bar()  
B.foo()
```

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

## הורשה (2)

```
public class A {  
    private void foo() {  
        System.out.println("A.foo()");  
    }  
  
    public void bar() {  
        System.out.println("A.bar()");  
        foo();  
    }  
}
```

```
public class B extends A {  
    public void foo() {  
        System.out.println("B.foo()");  
    }  
  
    public static void main(String[]  
        args) {  
        A a = new B();  
        a.bar();  
    }  
}
```

הפלט:  
A.bar()  
A.foo()

האם הקוד מתקמפל? אם לא, למה?  
אם כן, האם יש שגיאת ריצה? אם יש, למה?  
אחרת, מה הפלט?

# הורשה ובנאים

```
public class A {
    String bar = "A.bar";

    A() { foo(); }

    public void foo() {
        System.out.println("A.foo(): bar = " +
            bar);
    }
}

public class B extends A {
    String bar = "B.bar";

    B() { foo(); }

    public void foo() {
        System.out.println("B.foo(): bar = " +
            bar);
    }
}
```

```
public class C {
    public static void main(String[]
        args) {
        A a = new B();
        System.out.println("a.bar = "
            + a.bar);
        a.foo();
    }
}
```

הפלט:

```
B.foo(): bar = null
B.foo(): bar = B.bar
a.bar = A.bar
B.foo(): bar = B.bar
```

## הורשה ובנאים (2)

```
public class A {
    protected B b = new B();
    public A() { System.out.println("in A: no args."); }
    public A(String s) { System.out.println("in A: s = " + s); }
}

public class B {
    public B() { System.out.println("in B: no args."); }
}

public class C extends A {
    protected B b;
    public C() { System.out.println("in C: no args."); }
    public C(String s) { System.out.println("in C: s = " + s); }
}

public class D {
    public static void main(String args[]) {
        C c = new C();
        A a = new C();
    }
}
```

הפלט:

```
in B: no args.
in A: no args.
in C: no args.
in B: no args.
in A: no args.
in C: no args.
```



# הורשה ובנאים (3)

```
public class A {
    protected B b = new B();
    public A() { System.out.println("in A: no args."); }
    public A(String s) { System.out.println("in A: s = " + s); }
}

public class B {
    public B() { System.out.println("in B: no args."); }
}

public class C extends A {
    protected B b;
    public C() { System.out.println("in C: no args."); }
    public C(String s) { System.out.println("in C: s = " + s); }
}

public class D {
    public static void main(String args[]) {
        C c = new C("c");
        A a = new C("a");
    }
}
```

הפלט:

```
in B: no args.
in A: no args.
in C: s = c
in B: no args.
in A: no args.
in C: s = a
```

# סדר הפעולות ביצירת אובייקט

- אתחול ערך דיפולטי לשדות.
- קריאה לבנאי של מחלקת האב (שגורר אותו סדר פעולות רקורסיבית).
- אתחול שדות לפי הערכים שהושמו להם בשורה שבה הם מוגדרים.
- ביצוע שאר הקוד של הבנאי.

# דריסה והעמסה של שירותים

```
public class A {  
    public float foo(float a, float b) throws IOException {  
    }  
}
```

```
public class B extends A {  
    ...  
}
```

אילו מהשירותים הבאים ניתן להגדיר ב-B?

1. `float foo(float a, float b){...}`
2. `public int foo(int a, int b) throws Exception{...}`
3. `public float foo(float a, float b) throws Exception{...}`
4. `public float foo(float p, float q) {...}`

# הורשה ודריסת שירותים

```
public class A {  
    public void foo() {...}  
}
```

האם אפשר לקרוא ל-foo של  
A מתוך B?

```
public class B extends A {  
    public void foo() {...}  
}
```

תשובה:  
דרך `super.foo()`

## הורשה ודריסת שירותים (2)

```
public class A {  
    public void foo() {...}  
}
```

האם אפשר לקרוא ל-foo של  
A מתוך C?

```
public class B extends A {  
    public void foo() {...}  
}
```

**תשובה:**  
אי אפשר,  
`super.super.foo()`  
- לא חוקי

```
public class C extends B {  
    public void foo() {...}  
}
```

# מחלקות פנימיות

```
public class Test {  
    public int a = 0;  
    private int b = 1;
```

אילו משתנים מ-a-e נגישים מהשורה  
המסומנת?

```
    public void foo(final int c) {  
        int d = 2;
```

תשובה: כולם חוץ מ-d

```
        class InnerTest {  
            private void bar(int e) {
```

```
                
```

```
            }
```

```
        }  
        d = 3;
```

```
        a = 3;
```

```
    }
```

```
}
```

# מחלקות פנימיות - סיכום

Type	Scope	Inner	Interface	Fields access
<b>Static nested</b>	member	no	yes	Only static
<b>Inner non-static</b>	member	yes	no	Static and non-static
<b>local</b>	Local scope	yes	no	Effectively final local variables or parameters that are accessible in the scope of the block
<b>anonymous</b>	Only the point where it is defined	yes	no	Effectively final local variables or parameters that are accessible in the scope of the block

# enum

```
public class EnumTest {
```

```
    public enum Day {  
        SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
        THURSDAY, FRIDAY, SATURDAY;  
    }
```

```
        private Day(){
```

```
        }
```

```
    }
```

```
    Day day;
```

```
    public EnumTest(Day day) {  
        this.day = day;
```

```
    }
```

**All enums implicitly extend java.lang.Enum**  
**An enum cannot extend anything else.**

**The constructor for an enum type must be package-private or private access. You cannot invoke an enum constructor yourself.**

**fixed set of constants**



# enum

```
public void tellItLikeItIs() {
    switch (day) {
        case MONDAY: System.out.println("Mondays are bad.");
                       break;

        case FRIDAY: System.out.println("Fridays are better.");
                       break;

        case SATURDAY:
        case SUNDAY: System.out.println("Weekends are best.");
                       break;

        default:      System.out.println("Midweek days are so-so.");
                       break;
    }
}

public static void main(String[] args) {
    EnumTest firstDay = new EnumTest(Day.MONDAY);
    firstDay.tellItLikeItIs();
    for (Day d : Day.values()) {
        System.out.println(d);
    }
}
```

## Output:

Mondays are bad.  
SUNDAY  
MONDAY  
TUESDAY  
WEDNESDAY  
THURSDAY  
FRIDAY  
SATURDAY

static values method  
that returns an array  
containing all of the  
values of the enum in  
the order they are  
declared

# אוספים גנריים

```
public static void func(HashSet<String> set){  
    for (String s : set){  
        System.out.println(s);  
    }  
}
```

```
public static void main(String[] args){  
    HashSet<String> mySet = new HashSet<String>();  
    mySet.add("abc");  
    mySet.add("dce");  
    func(mySet);  
}
```

ניתן (ואפילו רצוי) לכתוב גם:  
`new HashSet<>();`

# אוספים גנריים

```
public static void func(HashSet<String> set){
    for (String s : set){
        System.out.println(s);
    }
}

public static void main(String[] args){
    HashSet<String> mySet = new HashSet<String>();
    mySet.add("abc");
    mySet.add("dce");
    func(mySet);
}
```

האם אנחנו חייבים להצהיר על טיפוס סטטי שהוא HashSet?  
בד"כ נשתמש בטיפוס הכללי יותר Set אלא אם כן אנחנו נדרשים ספציפית ל HashSet. למשל במקרים הבאים:

1. אנחנו רוצים להפעיל מתודה שיש ל HashSet אך לא ל Set (יש כזו בכלל?)
2. אנחנו משתמשים בשירות שדורש לקבל רק HashSet ולא Set.

# אוספים גנריים

```
public static void func(HashSet<String> set){
    for (String s : set){
        System.out.println(s);
    }
}

public static void main(String[] args){
    HashSet<String> mySet = new HashSet<String>();
    mySet.add("abc");
    mySet.add("dce");
    func(mySet);
}
```

מדוע הפונקציה דורשת לקבל HashSet? בד"כ נשתמש בטיפוס כמה שיותר כללי. האם נוכל לשלוח לפה כל Set? ע"מ המימוש שלה, אין סיבה שלא. למען האמת, נוכל לשלוח אפילו Collection.

# אוספים גנריים

```
public static void func(Collection<String> set){  
    for (String s : set){  
        System.out.println(s);  
    }  
}
```

```
public static void main(String[] args){  
    Set<String> mySet = new HashSet<>();  
    mySet.add("abc");  
    mySet.add("dce");  
    func(mySet);  
}
```

האם יש עוד משהו שנוכל לשפר בקוד?  
נשים לב כי המימוש של `func` לא מחייב אותנו לקבל אוסף של  
מחרוזות. הדרישה היחידה היא שאברי האוסף יממשו את  
`toString`, מה שמובטח לכל אובייקט ב `Java`.

# אוספים גנריים

```
public static void func(Collection<?> set){
    for (Object s : set){
        System.out.println(s);
    }
}
```

```
public static void main(String[] args){
    Set<String> mySet = new HashSet<>();
    mySet.add("abc");
    mySet.add("dce");
    func(mySet);
}
```

נשתמש ב `<?>` ע"מ לאפשר שימוש באוספים של כל טיפוס אפשרי.  
שימו לב, להגדיר את `set` מטיפוס `Collection<Object>` לא ישיג את אותה המטרה, כיוון שאז נוכל להפעיל את הפונקציה הזו רק עם אובייקט מטיפוס סטטי `Collection<Object>`

# אוספים גנריים

```
public static void func(Collection<?> set){
    for (Object s : set){
        System.out.println(s);
    }
}
```

```
public static void main(String[] args){
    Set<String> mySet = new HashSet<>();
    mySet.add("abc");
    mySet.add("dce");
    func(mySet);
}
```

נשתמש ב `<?>` ע"מ לאפשר שימוש באוספים של כל טיפוס אפשרי.  
שימו לב, להגדיר את `set` מטיפוס `Collection<Object>` לא ישיג את אותה המטרה, כיוון שאז נוכל להפעיל את הפונקציה הזו רק עם אובייקט מטיפוס סטטי `Collection<Object>`

# אוספים גנריים

```
public static void func(Collection<? extends Rectangle> set){  
    for (Rectangle s : set){  
        System.out.println(s.getArea());  
    }  
}
```

```
public static void main(String[] args){  
    Set<Rectangle> mySet = new HashSet<>();  
    mySet.add(new Rectangle(5, 6));  
    mySet.add(new Rectangle(1,2));  
    func(mySet);  
}
```