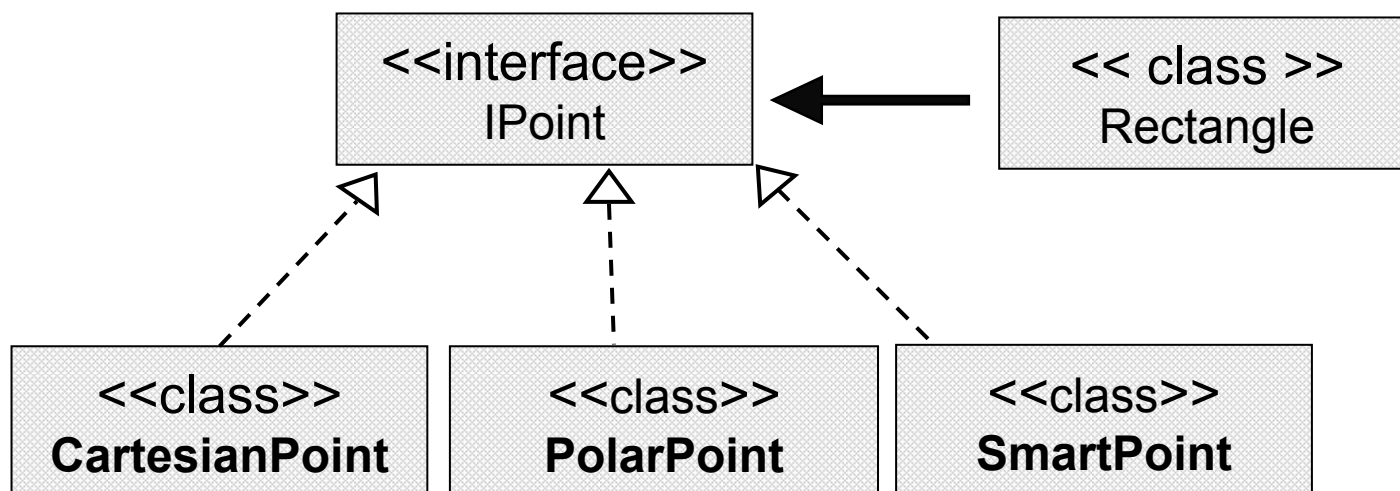


תוכנה 1 בשפת Java תרגול מספר 7: הורשה

בית הספר למדעי המחשב
אוניברסיטת תל אביב

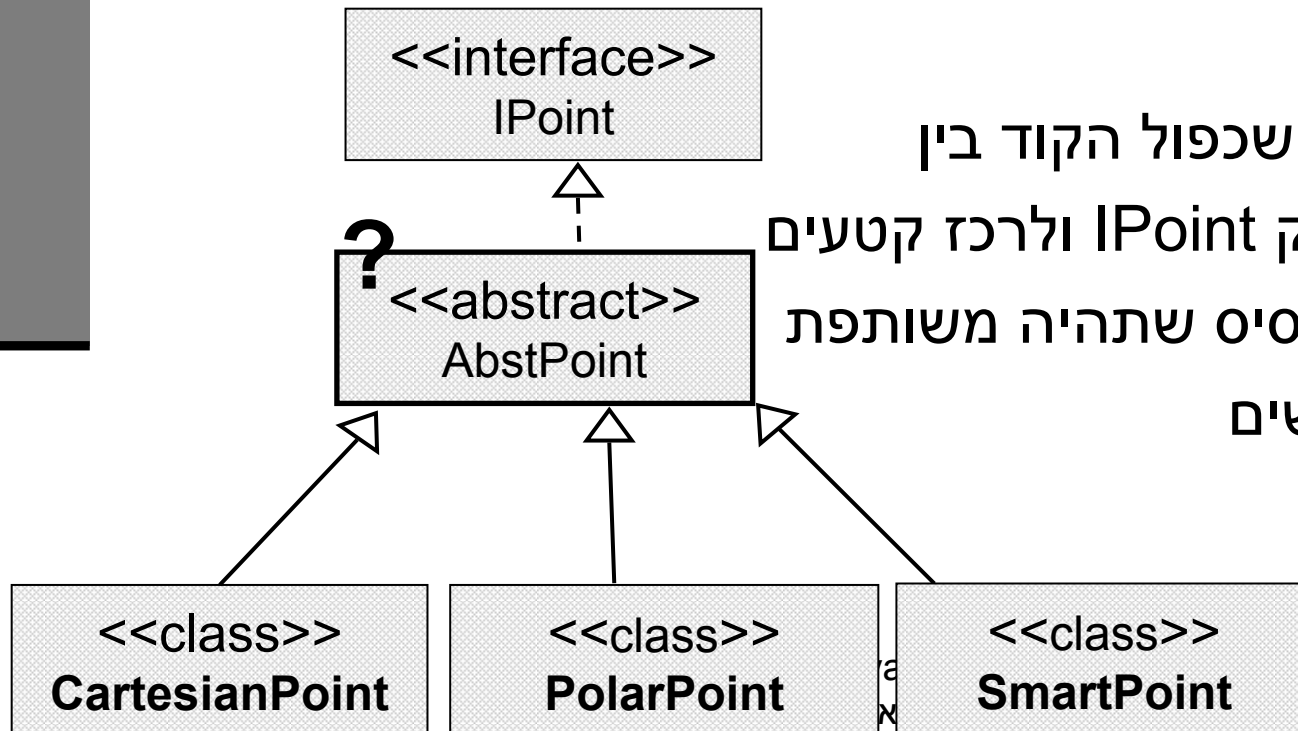
צד הלקוח

- בהרצאה ראינו את המנשק IPoint, והצגנו 3 מימושים שונים עבורו
- ראינו כי **לקוחות** התלויים במנשק IPoint בלבד, ולא מכירים את המחלקות המממשות **אדישים** לשינויים עתידיים בקוד הספק
- שימוש **במנשקים** חוסך **שכפול קוד לקוח**, בכך שאותו קטע קוד עובד בצורה נכונה עם מגוון ספקים (פולימורפיזם)



צד הספק

- בהרצאה האחרונה היכרנו את מנגנון ההורשה אשר חוסך שכפול קוד בצד הספק
- ע"י הורשה מקבלת מחלקה את קטע הקוד בירושה במקום לחזור עליו. שני הספקים חולקים אותו הקוד



- ננסה לזהות את שכפול הקוד בין 3 מממשי המנשק IPoint ולרכז קטעים אלה במחלקת בסיס שתהיה משותפת לשלושת המימושים

מחלקות מופשטות



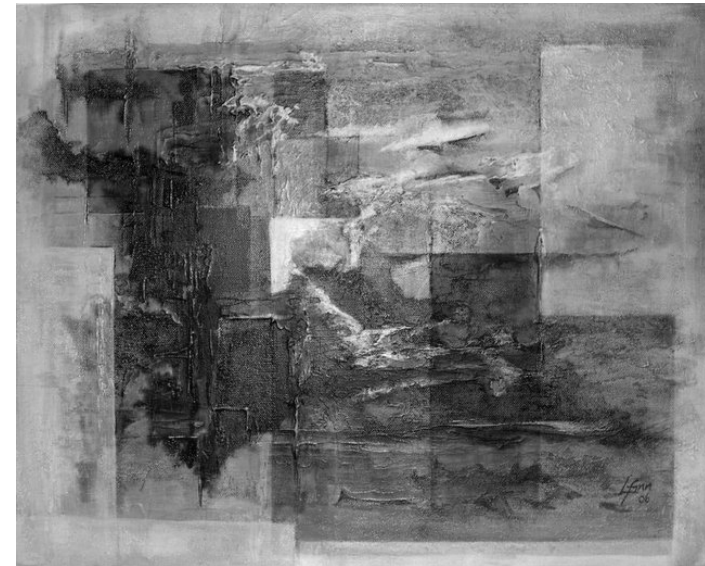
- מחלקה מופשטת מוגדרת ע"י המלה השמורה abstract
- לא ניתן ליצור מופע של מחלקה מופשטת (בדומה למנשק)
- יכולה לממש מנשק אך לא לממש את כל השירותים המוגדרים בו
- זהו מנגנון מועיל להימנע משכפול קוד במחלקות יורשות

מחלקות מופשטות - דוגמא

■ מחלקה פשוטה:

```
public abstract class A {  
    public void f() {  
        System.out.println("A.f!");  
    }  
  
    public void g();  
}  
  
A a = new A();  
  
public class B extends A {  
    public void g() {  
        System.out.println("B.g!");  
    }  
}
```

```
A a = new B();
```



CartesianPoint

```
private double x;  
private double y;
```

```
public CartesianPoint(double x, double y) {  
    this.x = x;  
    this.y = y;  
}
```

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

```
public double y() { return y;}
```

```
public double rho() { return Math.sqrt(x*x + y*y); }
```

```
public double theta() { return Math.atan2(y,x); }
```

PolarPoint

```
private double r;  
private double theta;
```

```
public PolarPoint(double r, double theta) {  
    this.r = r;  
    this.theta = theta;  
}
```

```
public double x() { return r * Math.cos(theta); }
```

```
public double y() { return r * Math.sin(theta); }
```

```
public double rho() { return r; }
```

```
public double theta() { return theta; }
```

קשה לראות דמיון בין מימושי המתודות במקרה זה.
כל 4 המתודות בסיסיות ויש להן קשר הדוק לייצוג שנבחר ל**שדות**

CartesianPoint

```
public void rotate(double angle) {  
    double currentTheta = Math.atan2(y,x);  
    double currentRho = rho();  
  
    x = currentRho * Math.cos(currentTheta+angle);  
    y = currentRho * Math.sin(currentTheta+angle);  
}
```

PolarPoint

```
public void rotate(double angle) {  
    theta += angle;  
}
```

```
public void translate(double dx, double dy) {  
    x += dx;  
    y += dy;  
}
```

```
public void translate(double dx, double dy) {  
    double newX = x() + dx;  
    double newY = y() + dy;  
    r = Math.sqrt(newX*newX + newY*newY);  
    theta = Math.atan2(newY, newX);  
}
```

גם כאן קשה לראות דמיון בין מימושי המתודות,
למימושים קשר הדוק לייצוג שנבחר לשדות

CartesianPoint

```
public double distance(IPoint other) {  
    return Math.sqrt((x-other.x()) * (x-other.x()) +  
        (y-other.y())*(y-other.y()));  
}
```

PolarPoint

```
public double distance(IPoint other) {  
    double deltaX = x()-other.x();  
    double deltaY = y()-other.y();  
  
    return Math.sqrt(deltaX*deltaX +  
        deltaY*deltaY);  
}
```

הקוד דומה אבל לא זהה, נראה מה ניתן לעשות...

ננסה לשכתב את **CartesianPoint** ע"י הוספת משתני העזר ΔX ו- ΔY

CartesianPoint

```
public double distance(IPoint other) {  
    double deltaX = x-other.x();  
    double deltaY = y-other.y();  
  
    return Math.sqrt((x-other.x()) * (x-other.x()) +  
                    (y-other.y()) * (y-other.y()));  
}
```

PolarPoint

```
public double distance(IPoint other) {  
    double deltaX = x()-other.x();  
    double deltaY = y()-other.y();  
  
    return Math.sqrt(deltaX*deltaX +  
                    deltaY*deltaY);  
}
```

CartesianPoint

```
public double distance(IPoint other) {  
    double deltaX = x-other.x();  
    double deltaY = y-other.y();  
  
    return Math.sqrt(deltaX * deltaX +  
        (deltaY * deltaY));  
}
```

PolarPoint

```
public double distance(IPoint other) {  
    double deltaX = x()-other.x();  
    double deltaY = y()-other.y();  
  
    return Math.sqrt(deltaX*deltaX +  
        deltaY*deltaY);  
}
```

נשאר הבדל אחד

נחליף את x להיות x() –
במאזן ביצועים לעומת כלליות נעדיף תמיד את הכלליות

CartesianPoint

```
public double distance(IPoint other) {  
    double deltaX = x()-other.x();  
    double deltaY = y()-other.y();  
  
    return Math.sqrt(deltaX * deltaX +  
        (deltaY * deltaY ));  
}
```

PolarPoint

```
public double distance(IPoint other) {  
    double deltaX = x()-other.x();  
    double deltaY = y()-other.y();  
  
    return Math.sqrt(deltaX*deltaX +  
        deltaY*deltaY);  
}
```

שתי המתודות זהות לחלוטין.
ניתן להעביר את המתודה למחלקה AbstPoint
ולמחוק אותה מהמחלקות CartesianPoint ו-PolarPoint

CartesianPoint

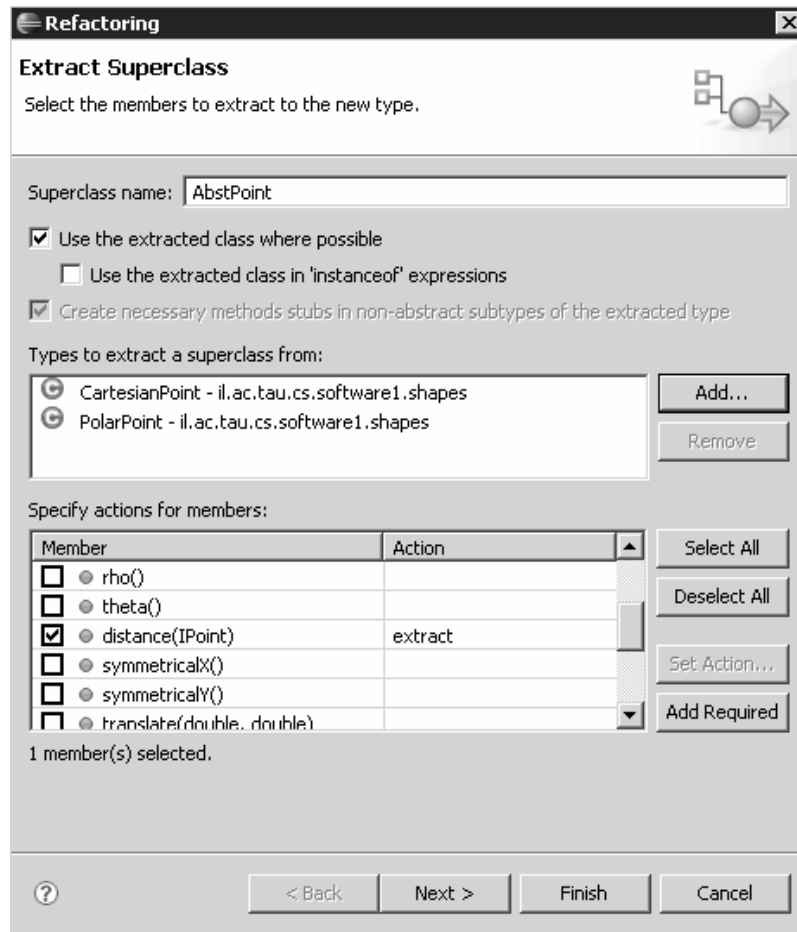
```
public String toString(){  
    return "(x=" + x + ", y=" + y +  
        ", r=" + rho() + ", theta=" + theta() + ")";  
}
```

PolarPoint

```
public String toString() {  
    return "(x=" + x() + ", y=" + y() +  
        ", r=" + r + ", theta=" + theta + ")";  
}
```

תהליך דומה ניתן גם לבצע עבור toString

Extract Superclass Refactoring



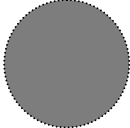
■ ניתן לבצע תהליך זה בצורה
אוטומטית ע"י שכתוב מבני
(Refactoring) שנקרא:
Extract Superclass

■ הגרסה ב- Eclipse עוד לא
"מושלמת"

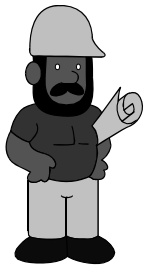


אתחולים ובנאים

נעבוד עם דוגמא



```
public class Object {  
    public Object() {}  
}
```



```
public class Manager extends Employee{  
    private String department;  
    public Manager(String n, String d) {  
        super(n);  
        department = d;  
    }  
}
```

```
public class Employee extends Object {  
    private String name;  
    private double salary = 15000.00;  
    private Date birthDate;  
  
    public Employee(String n, Date DoB) {  
        // implicit super();  
        name = n;  
        birthDate = DoB;  
    }  
    public Employee(String n) {  
        this(n, null);  
    }  
}
```

מה הסדר ביצירת מופע של מחלקה?

■ שלב ראשון: הקצאת זיכרון לשדות העצם והצבת ערכי ברירת מחדל

```
Manager m = new Manager("Joe Smith", "Sales");
```



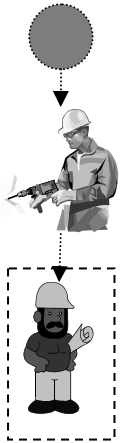
```
(String)Name  
(double)Salary  
(Date)Birth Date  
(String)Department
```


מה הסדר ביצירת מופע של מחלקה?

■ שלב שני: נקרא הבנאי הנוכחי והאלגוריתם הבא מופעל:

1. Bind constructor parameters.
2. If explicit this(), call recursively, and then skip to Step 5.
3. Call recursively the implicit or explicit super(...), except for Object because Object has no parent class.
4. Execute the explicit instance variable initializers.
5. Execute the body of the current constructor.

בואו נעקוב אחרי יצירת האובייקט



■ Basic initialization

- Allocate memory for the complete Manager object
- Initialize all instance variables to their default values

■ Call constructor: **Manager("Joe Smith", "Sales")**

- Bind constructor parameters: n="Joe Smith", d="Sales"
- No explicit this() call
- Call super(n) for Employee(String)
 - Bind constructor parameters: n="Joe Smith"
 - Call this(n, null) for Employee(String, Date)
 - Bind constructor parameters: n="Joe Smith", DoB=null
 - No explicit this() call
 - Call super() for Object()
 - No binding necessary
 - No this() call
 - No super() call (Object is the root)
 - No explicit variable initialization for Object
 - No method body to call
 - Initialize explicit Employee variables: salary=15000.00;
 - Execute body: name="Joe Smith"; date=null;
 - Steps skipped
 - Execute body: No body in Employee(String)
- No explicit initializers for Manager
- Execute body: department="Sales"

```
public class Manager extends Employee
{
    private String department;
    public Manager(String n, String d) {
        super(n);
        department = d;
    }
}
```

```
public class Employee extends Object {
    private String name;
    private double salary = 15000.00;
    private Date birthDate;

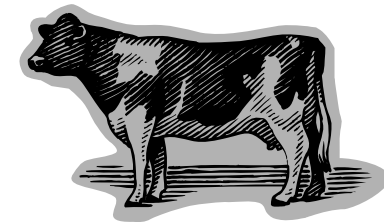
    public Employee(String n, Date DoB) {
        // implicit super();
        name = n;
        birthDate = DoB;
    }
    public Employee(String n) {
        this(n, null);
    }
}
```

18

1. Bind parameters.
2. explicit this(), goto 5.
3. super(),
4. explicit var. init.
5. Execute body

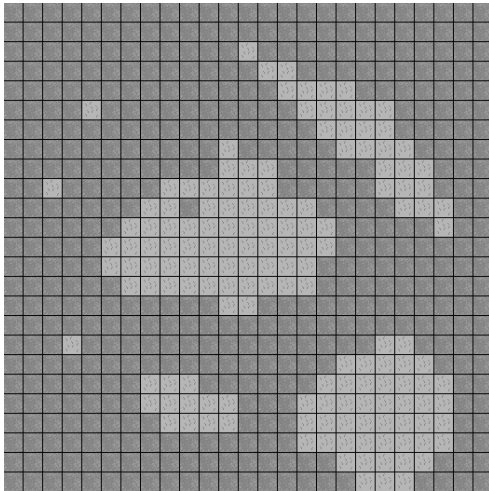
Exercise 7

טורפים ואוכלי עשב

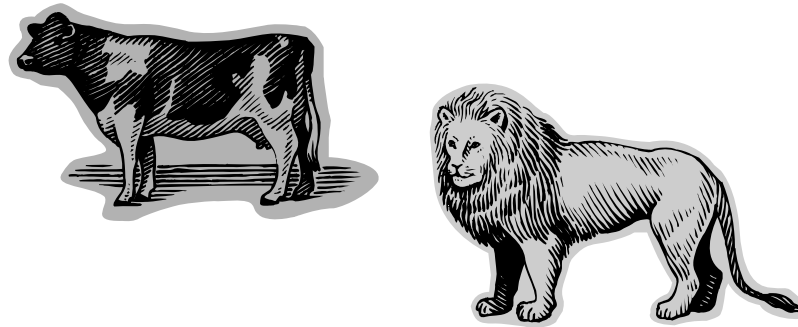


הפרויקט בגדול

■ שמות עצם בהגדרה

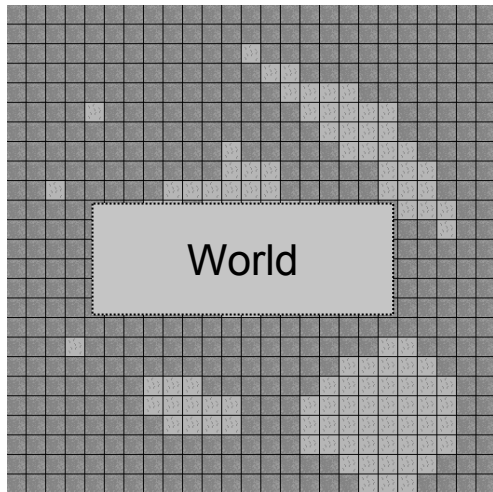


העולם



חיה – אוכלי עשב וטורפים

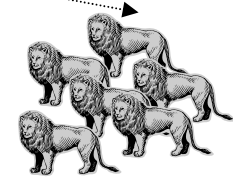
המחלקות



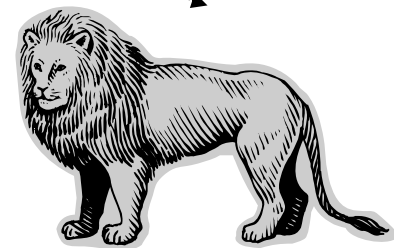
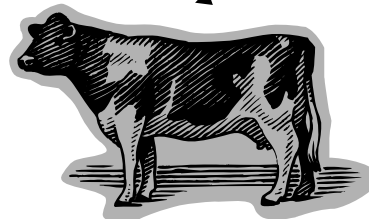
PatchState

Action

<<Interface>>
Species



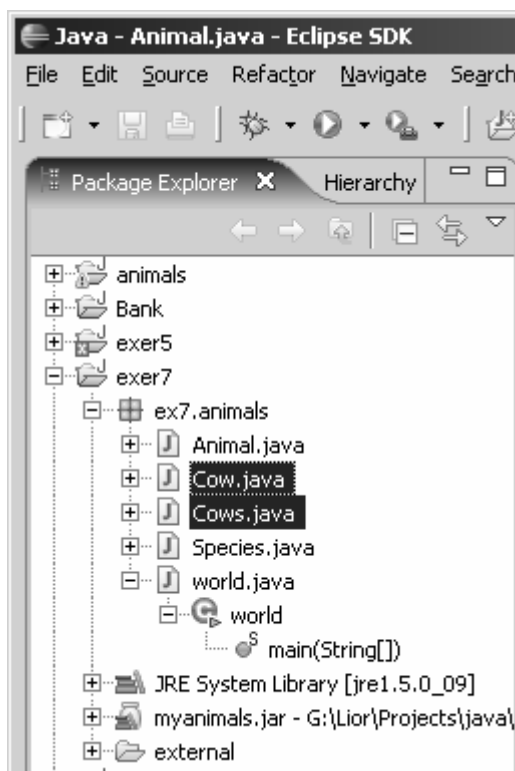
<<Interface>>
Animal



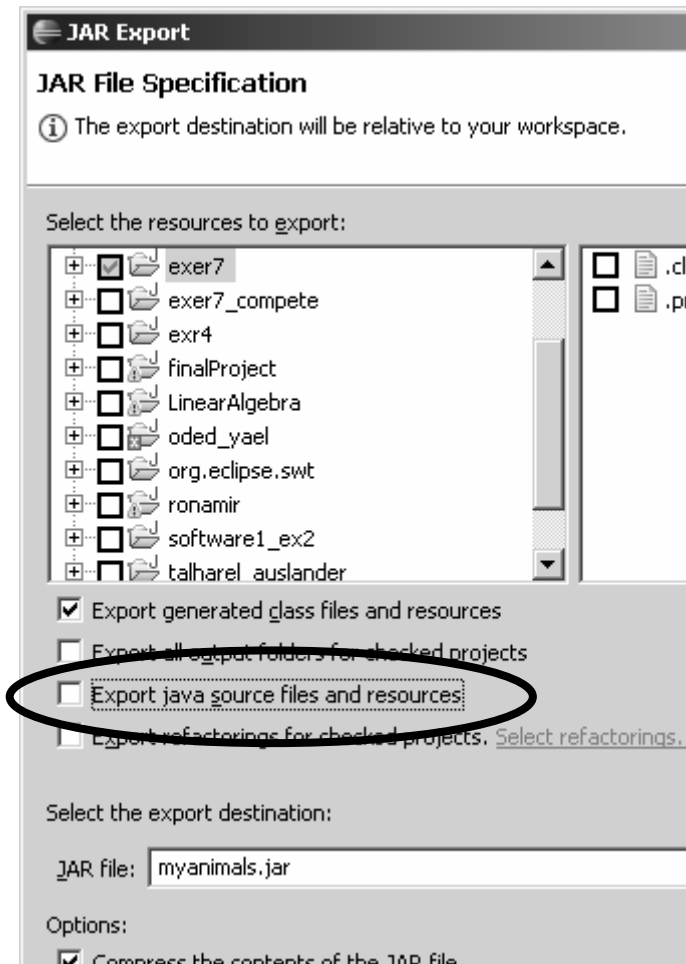
החלפת קבצים בינאריים

1. השלימו את הפרויקט לפי ההוראות עד סעיף 4

2. בחרו ב- Package Explorer את המחלקות שאתם רוצים להעביר לסטודנט אחר



החלפת קבצים בינאריים



1. השלימו את הפרויקט לפי ההוראות
סעיף 4

2. בחרו ב-Package Explorer את
המחלקות שאתם רוצים להעביר
לסטודנט אחר

3. לחצו כפתור ימני ו-export (או
מתפריט file)

4. ייצאו את הקבצים (ללא קבצי מקור)

החלפת קבצים בינאריים

■ העתיקו את קובץ ה-jar שקיבלתם
מסטודנט אחר אל ספריית הפרויקט
שלכם

■ בחרו ב-eclipse

■ Project → properties

■ הוסיפו את קובץ ה-jar המתאים

■ הוסיפו קריאות ליצירת המין החדש

■ הריצו את הפרויקט

