

# תוכנה 1

---

תרגול מספר 9:

תרגיל – חברת הייטק

## חברת הייטק

- בתרגיל זה נתרגל מספר נושאים אותם למדנו בשיעורים האחרונים:
  - עיצוב ובניית מודל המורכב ממחלקות לתיאור סביבה מסוימת
  - מנשקים, מחלקות מופשטות וירוושה
  - אוספים
- במסגרת התרגיל נכתוב תכנית לחישוב שכר בחברת הייטק המורכבת ממספר סוגים של עובדים.

## עצבו מחלקות לייצוג עובדים בחברה על פי המפרט הבא:

- בחברת הייטק מצליחה ישנם 3 סוגי עובדים:
  - תוכניתנים
  - בודקי תוכנה
  - מנהלים
- לכל עובד יש:
  - שם
  - מזהה מספרי
  - בוס (מסוג מנהל)
- כל עובד מקבל משכורת.
- לכל מנהל יש רשימה של עובדים אותם הוא מנהל.
- לכל תוכניתן יש שפת תכנות מועדפת (מתוך רשימה אפשרית)

## המשך המפרט:

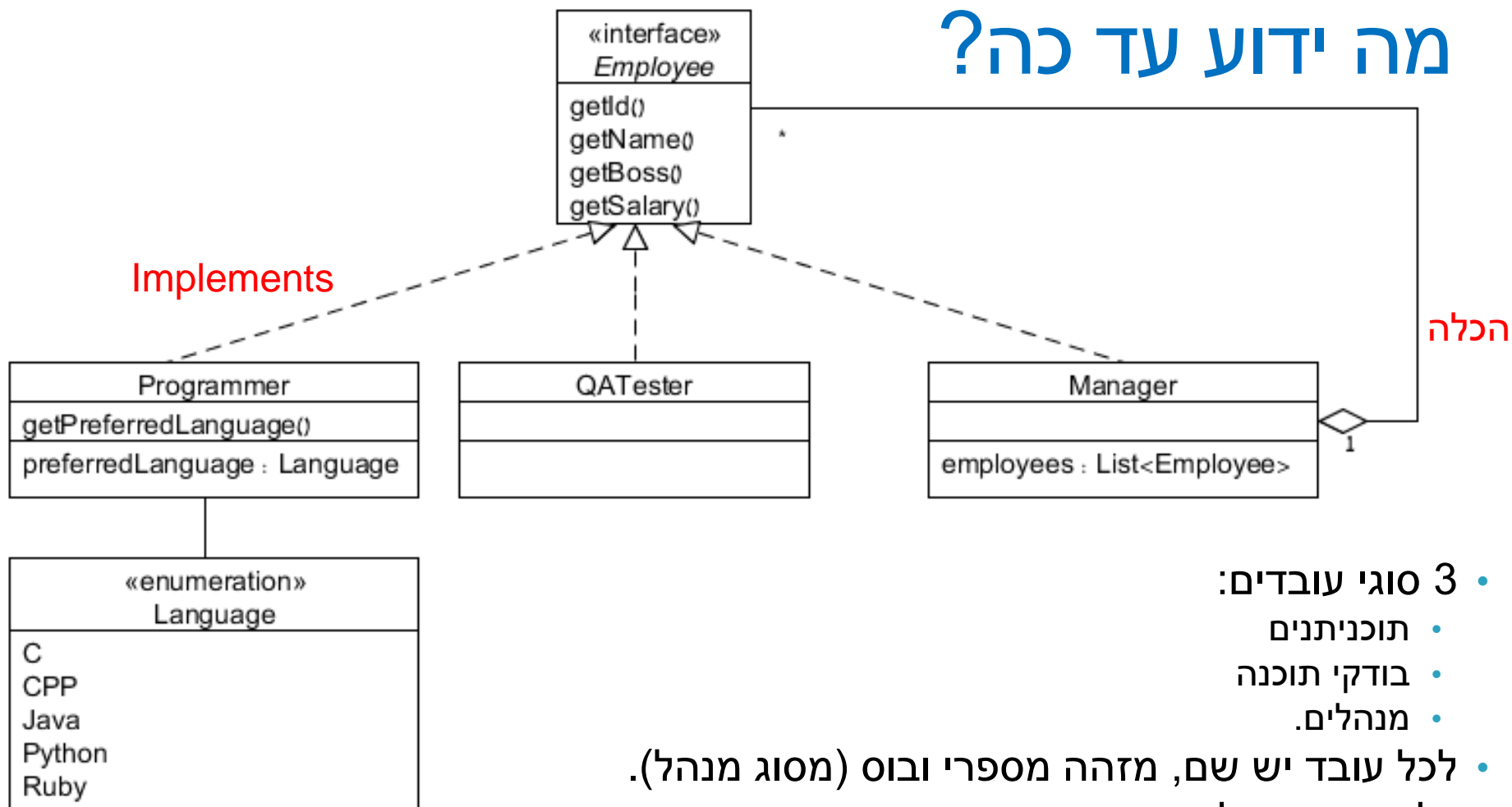
### • שכר:

- **תוכניתנים ובודקי תוכנה מקבלים שכר בסיס אישי**
- **בודקי תוכנה מקבלים גם בonus על כל באג שמצאו השבוע (בonus קבוע לכל הבודקים).**
- **מנהל מקבל שכר אשר נקבע כמספר העובדים שהוא מנהל ישירות \* פקטור אישי.**

נתחיל?

---

# מה ידוע עד כה?



## • 3 סוגי עובדים:

- תוכניתנים
- בודקי תוכנה
- מנהלים.
- לכל עובד יש שם, מזהה מספרי ובוס (מסוג מנהל).
- כל עובד מקבל משכורת.
- לכל מנהל יש רשימה של עובדים אותם הוא מנהל.
- לכל תוכניתן יש שפת תכנות מועדפת (מתוך רשימה אפשרית)

# המשך המפרט:

## • שכר:

- תוכניתנים ובודקי תוכנה מקבלים שכר בסיס אישי
- בודקי תוכנה מקבלים גם בונוס על כל באג שמצאו השבוע (בונוס קבוע לכל הבודקים).
- מנהל מקבל:
- שכר אשר נקבע כמספר העובדים שהוא מנהל ישירות \* פקטור אישי.

פונקציות

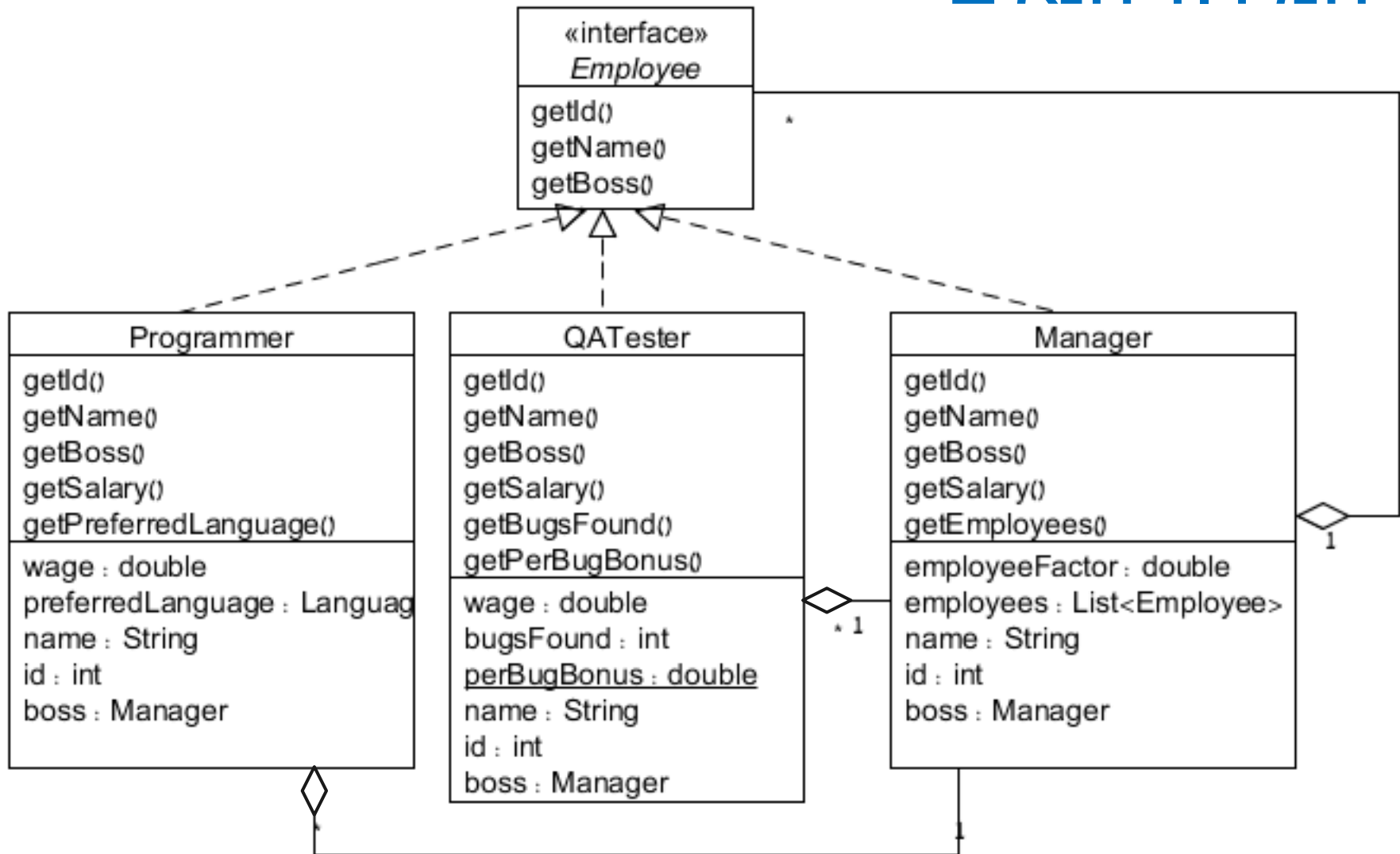
שדות

Programmer
getId() getName() getBoss() getSalary() getPreferredLanguage()
wage : double
preferredLanguage : Language name : String id : int boss : Manager

QATester
getId() getName() getBoss() getSalary() getBugsFound() getPerBugBonus()
wage : double
bugsFound : int
perBugBonus : double
name : String id : int boss : Manager

Manager
getId() getName() getBoss() getSalary() getEmployees()
employeeFactor : double
employees : List<Employee>
name : String id : int boss : Manager

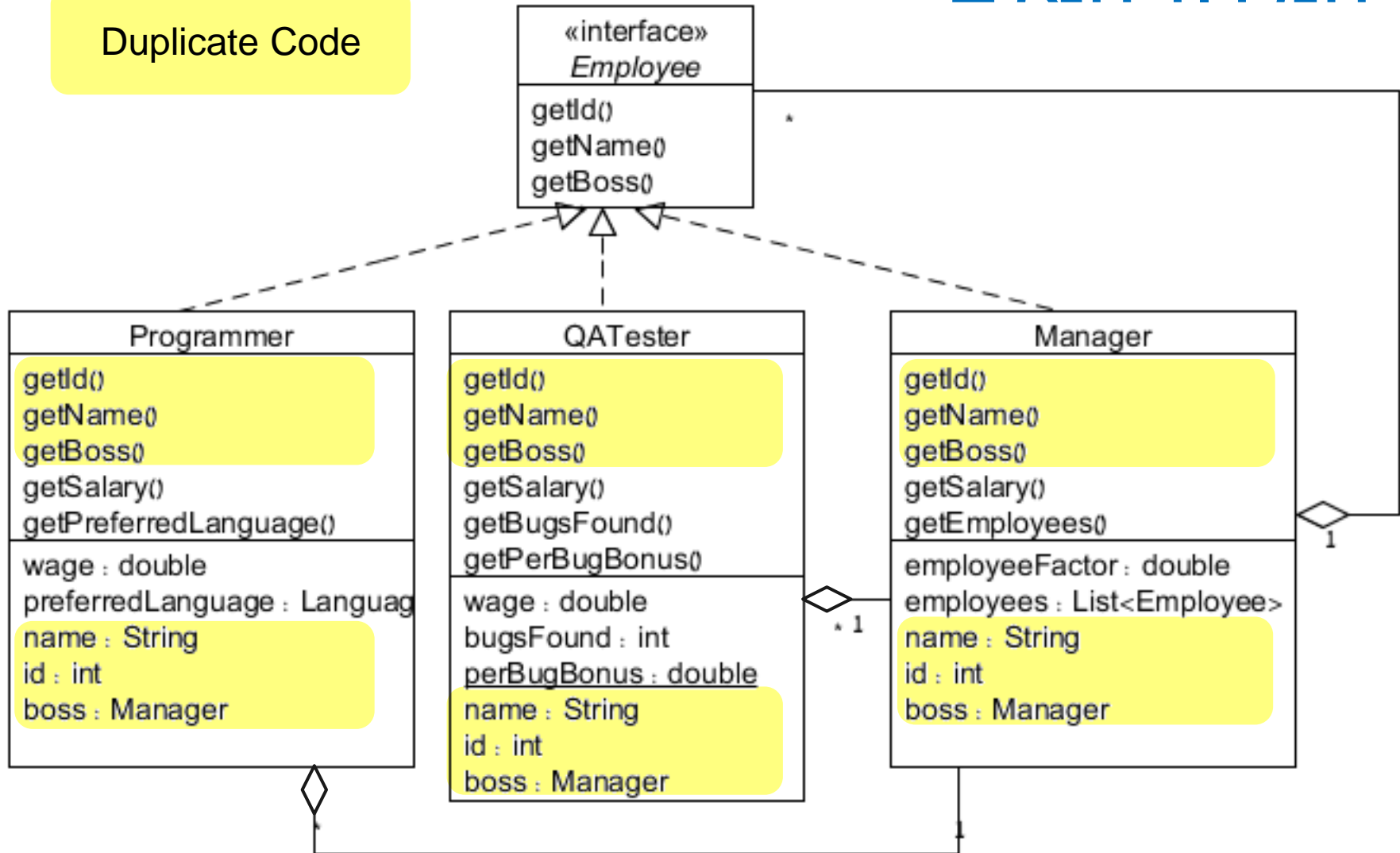
# המידול הנאיבי





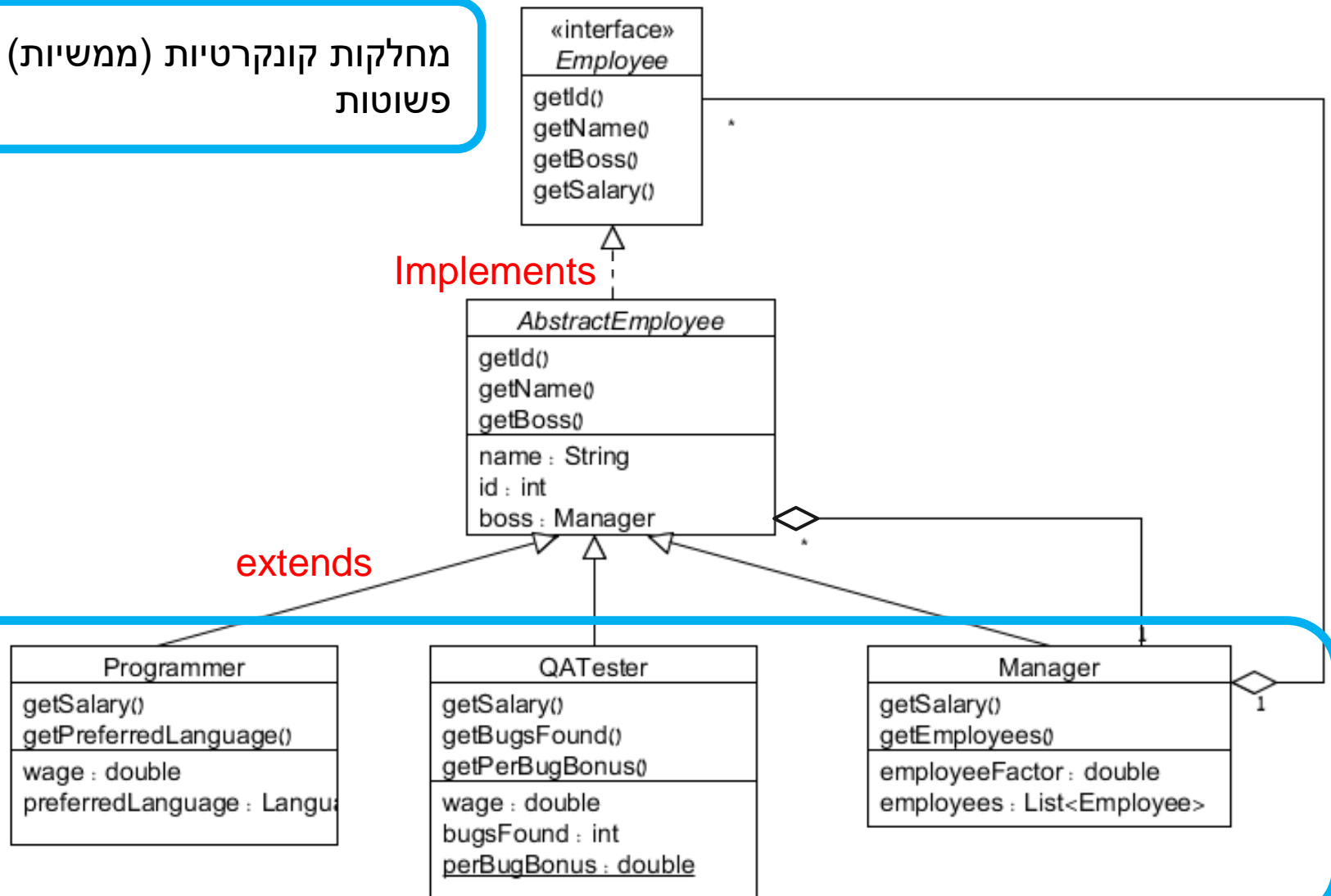
# המידול הנאיבי

Duplicate Code



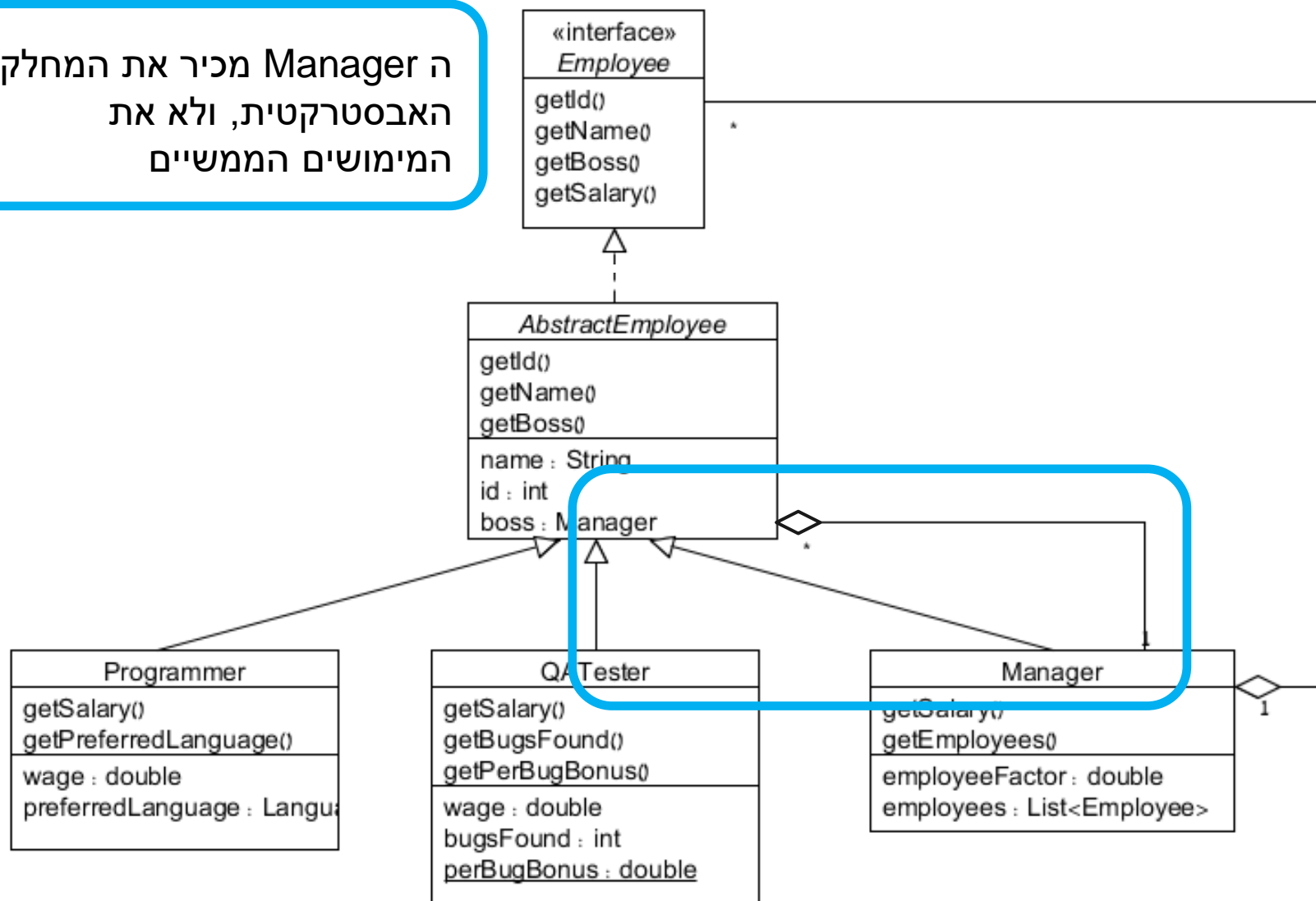
# שלב 1 – עובד אבסטרקטי

מחלקות קונקרטיות (ממשיות)  
פשוטות



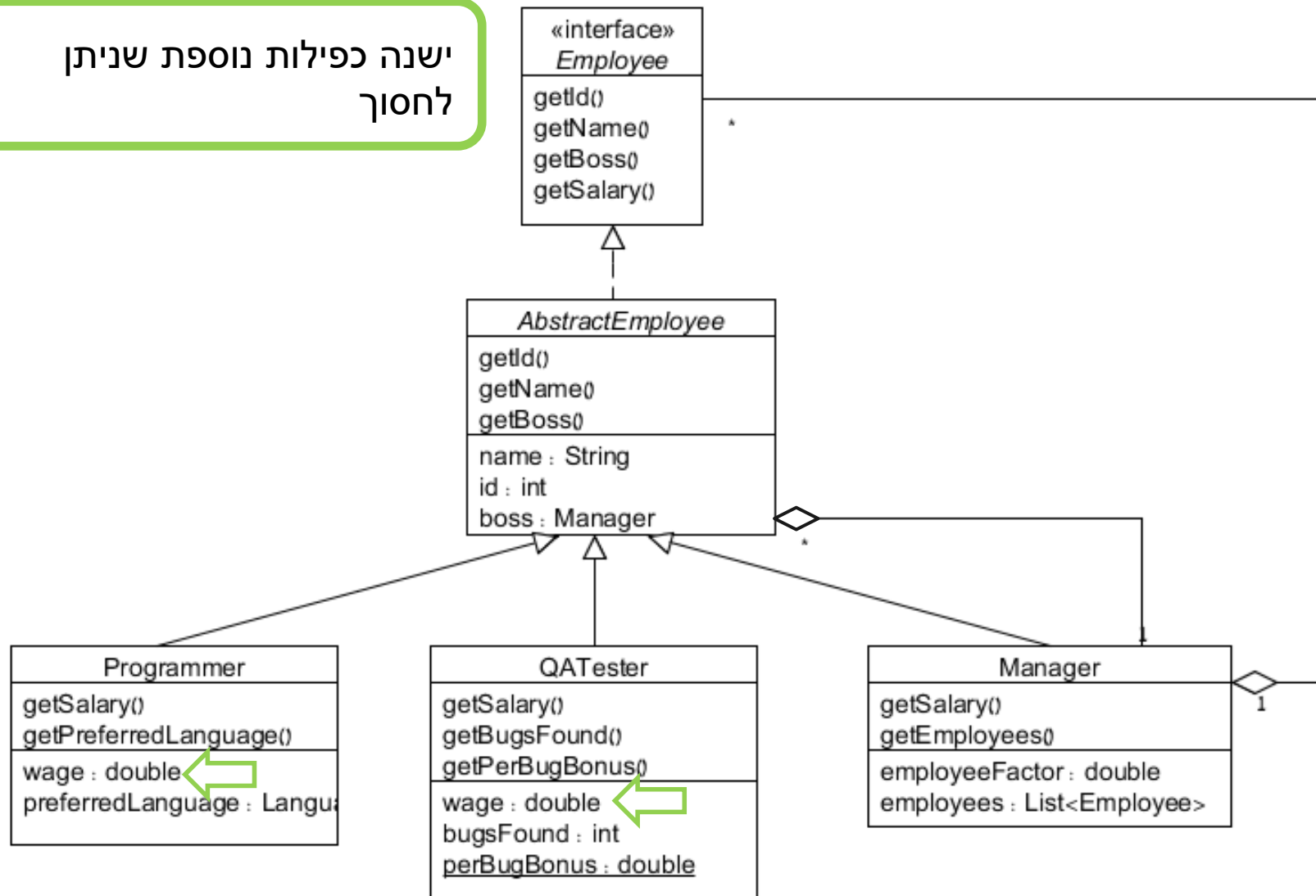
# שלב 1 – עובד אבסטרקטי

ה Manager מכיר את המחלקה האבסטרקטית, ולא את המימושים הממשיים

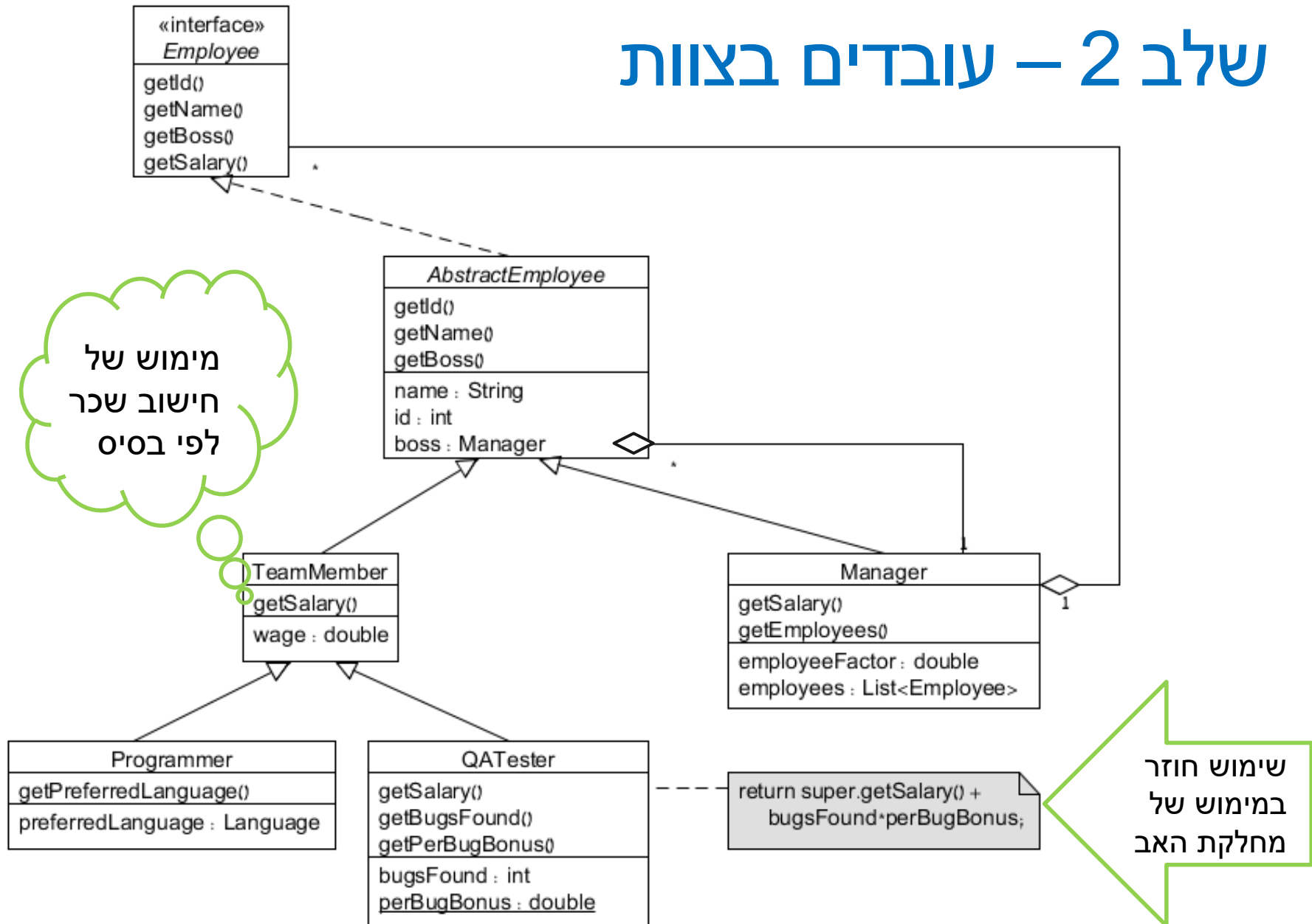


# שלב 1 – עובד אבסטרקטי

ישנה כפילות נוספת שניתן לחסוך

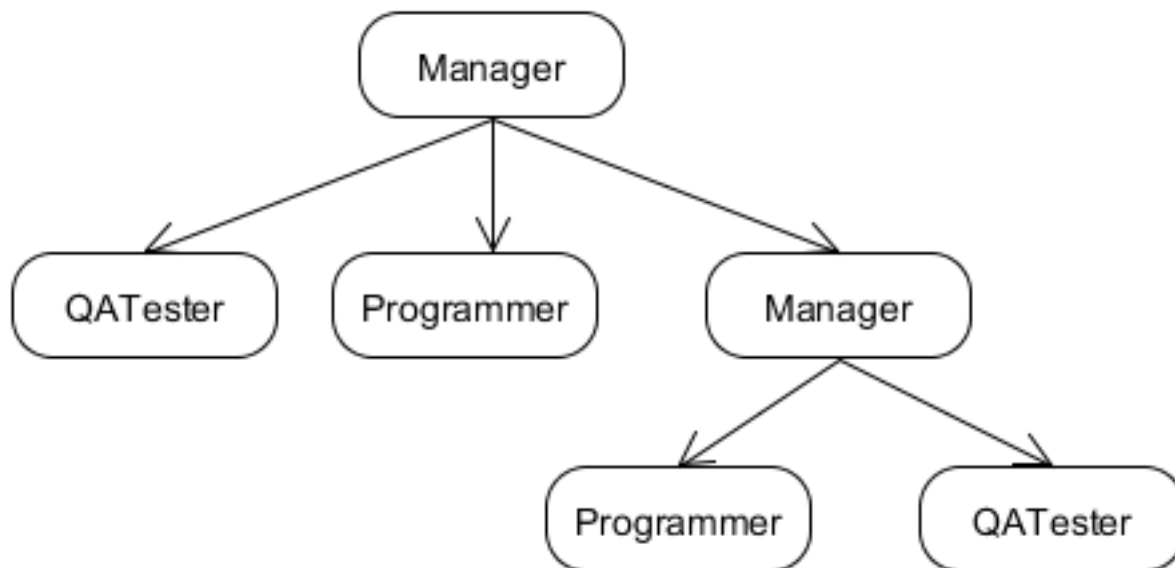


# שלב 2 – עובדים בצוות



## שלב 3 – plan ahead? (אופציונאלי)

- לפנינו מבנה היררכי (עץ)
- ייתכן שנרצה לעבור על המבנה בצורה אחידה



## שלב 3 – plan ahead? (אופציונאלי)

- לפנינו מבנה היררכי (עץ)
- ייתכן שנרצה לעבור על המבנה בצורה אחידה

```
public void getSalaries (Employee employee, List<Double> Salaries) {  
    Salaries.add(employee.getSalary());  
    if (employee instanceof Manager){  
        Manger manager = (Manager) employee;  
        List<Employee> employees = manager.getEmployees();  
        for (Employee e: employees){  
            getSalaries(e, Salaries);  
        }  
    }  
}
```

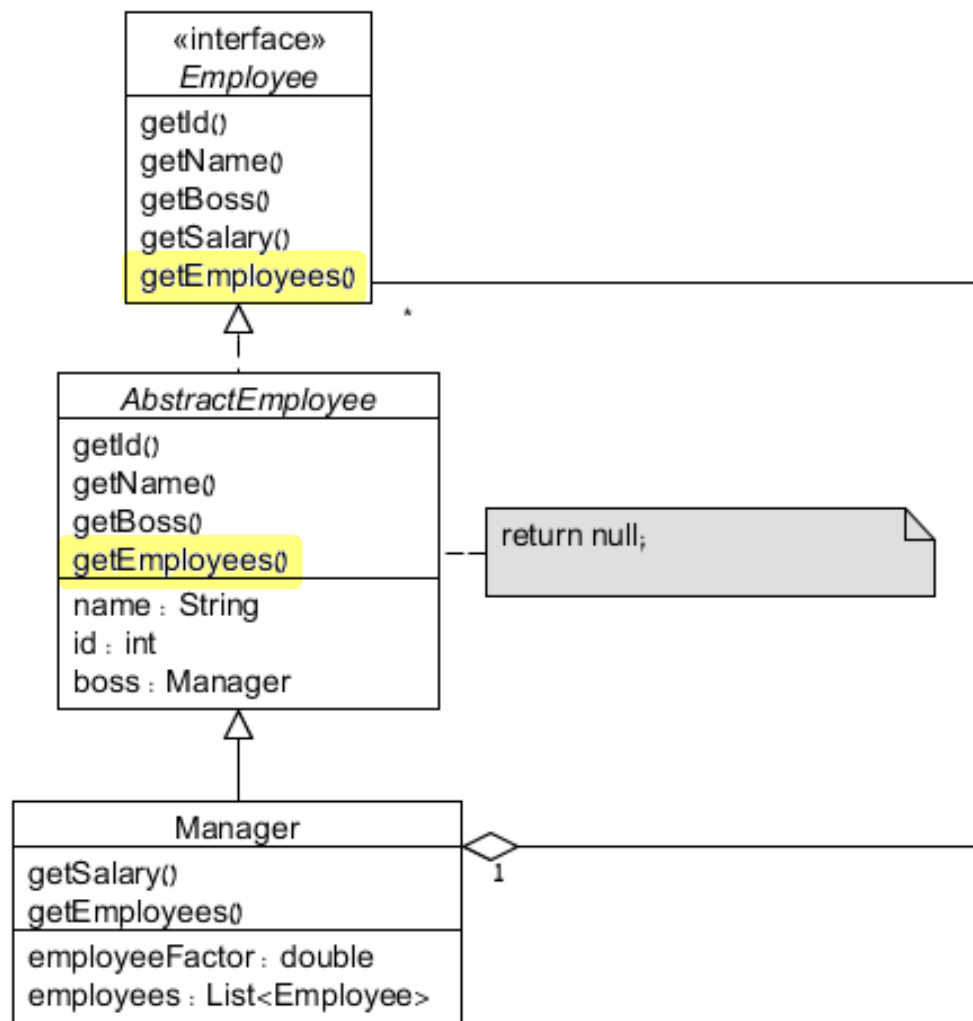
## שלב 3 – plan ahead? (אופציונאלי)

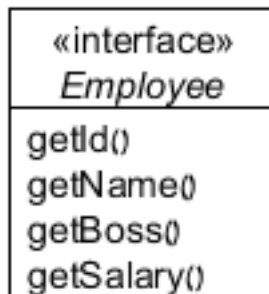
- לפנינו מבנה היררכי (עץ)
- ייתכן שנרצה לעבור על המבנה בצורה אחידה
- נבצע שינוי פשוט במחלקות כך שלכולם יהיה `getEmployees`, ואלה שאינם מנהלים יחזירו `null`

```
public void getSalaries (Employee employer, List<Double> Salaries) {  
    Salaries.add(employer.getSalary());  
    List<Employee> employees = employer.getEmployees();  
    if (employees != null){  
        for (Employee e: employees){  
            getSalaries(e, Salaries);  
        }  
    }  
}
```



# שלב 3 – plan ahead? (אופציונאלי)





## מה הלאה?

### • לכתוב קוד!

- נעבור רק על החלקים המרכזיים
- שאר הקוד באתר

```
public interface Employee {  
    public int getId();  
    public String getName();  
    public Manager getBoss();  
    public double getSalary();  
}
```



```

public abstract class AbstractEmployee implements Employee {
    private int id;
    private String name;
    private Manager boss;

    public AbstractEmployee(int id, String name, Manager boss) {
        this.id = id;
        this.name = name;
        this.boss = boss;
    }
    @Override
    public int getId() {
        return id;
    }
    @Override
    public String getName() {
        return name;
    }
    @Override
    public Manager getBoss() {
        return boss;
    }
}

```

<i>AbstractEmployee</i>
getId() getName() getBoss()
name : String id : int boss : Manager

# Enumerated types

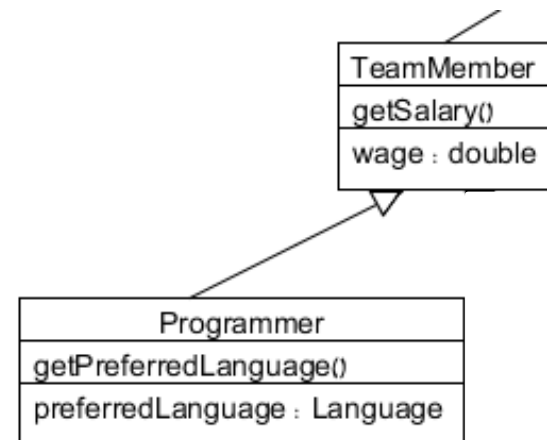
```
public enum Language {  
    C,  
    CPP,  
    Java,  
    Python,  
    Ruby;  
}
```

וריאציה יותר מתוחכמת,  
הכוללת הגדרת שדות ומתודות

```
public enum Language {  
    C("C"),  
    CPP("C++"),  
    Java("Java"),  
    Python("Python"),  
    Ruby("Ruby");  
  
    private final String displayName;  
  
    private Language(String name) {  
        displayName = name;  
    }  
  
    @Override  
    public String toString() {  
        return displayName;  
    }  
}
```

# Enumerated types - usage

```
public class Programmer extends TeamMember {  
  
    private Language preferredLanguage;  
  
    public Programmer(int id, String name, Manager boss, double wage,  
                      Language preferredLanguage) {  
        super(id, name, boss, wage);  
        this.preferredLanguage = preferredLanguage;  
    }  
  
    public Language getPreferredLanguage() {  
        return preferredLanguage;  
    }  
}
```

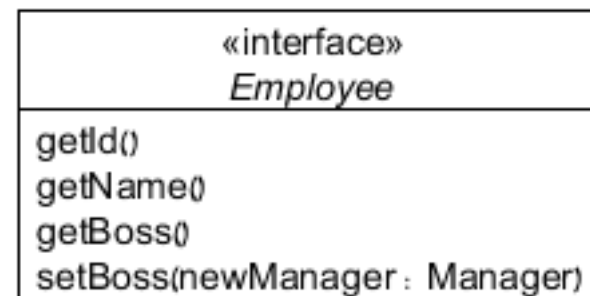


## פרטי מימוש...

- נרצה לוודא כי לעובד יש רק מנהל אחד.
- אין בעיה מצד העובד (משתנה יחיד למנהל)
- צריך לוודא שכאשר משנים מנהל מורידים את העובד מהרשימה המתאימה

```
public abstract class AbstractEmployee implements Employee {
    ...
    @Override
    public void setBoss(Manager newManager) {
        Employee oldBoss = getBoss();
        if(oldBoss != null)
            oldBoss.removeEmployee(this);

        this.boss = newManager;
        if(this.boss != null)
            this.boss.addEmployee(this);
    }
}
```



## פרטי מימוש...

- תמיכה ב-Hash
- (ניתן ל-eclipse לעשות את העבודה.)
- נסתמך על שדה ה-id.

```
public abstract class AbstractEmployee implements Employee {  
    ...  
    @Override  
    public int hashCode() {  
        final int prime = 31;  
        int result = 1;  
        result = prime * result + id;  
        return result;  
    }  
}
```

# פרטי מימוש...

- תמיכה ב-Collections
- (ניתן ל-eclipse לעשות את העבודה.)
- שוב, נסתמך על שדה ה-id.

```
public abstract class AbstractEmployee implements Employee {  
    ...  
    public boolean equals(Object obj) {  
        if (this == obj)  
            return true;  
        if (obj == null)  
            return false;  
        if (getClass() != obj.getClass())  
            return false;  
        AbstractEmployee other = (AbstractEmployee) obj;  
        if (id != other.id);  
            return false;  
        return true;  
    }  
}
```



# חישובי שכר

- למנהל חישוב שכר ייחודי

```
public class Manager extends AbstractEmployee {  
    @Override  
    public double getSalary() {  
        return employeeFactor * employees.size();  
    }  
}
```

# חישובי שכר

- חישוב שכר עפ"י שכר בסיס

```
public class TeamMember extends AbstractEmployee {  
  
    private double wage;  
  
    public TeamMember(int id, String name, Manager boss,  
                     double wage) {  
        super(id, name, boss);  
        this.wage = wage;  
    }  
  
    @Override  
    public double getSalary() {  
        return wage;  
    }  
}
```

# חישובי שכר

• חישוב שכר עפ"י שכר בסיס + בonus

```
public class QATester extends TeamMember {
    private static double PER_BUG_BONUS = 100.0;
    private int bugsFound = 0;

    public QATester(int id, String name, Manager boss, double wage) {
        super(id, name, boss, wage);
    }

    public void incrementBugs() { this.bugsFound++; }
    public int getBugsFound() { return bugsFound; }

    @Override
    public double getSalary() {
        return super.getSalary() + getBugsFound() * PER_BUG_BONUS;
    }
}
```



## עוד דרישות:

- כתבו תכנית המייצרת אובייקטים של עובדים עם נתונים אקראיים ושומרת אותם בשלוש רמות היררכיות לפי הפירוט הבא:
  - בראש ההיררכיה נמצא המנכ"ל שהינו מנהל
  - מתחתיו בהיררכיה יש 5 מנהלים
  - מתחת לכל מנהל מצויים בהיררכיה 10 תכניתנים או בודקי תוכנה (בהסתברות שווה).
- לאחר מכן, התוכנית תדפיס את פרטי 3 העובדים עם המשכורת הגבוהה ביותר בכל רמה היררכית.

# דוגמא לפלט:

<b>CEO:</b>				
ID: 1	Name: Taylor Zuckerberg	Boss: None	Salary: 49740.43	Employees: 5
<b>Managers:</b>				
ID: 13	Name: Kate Hewlett	Boss: Taylor Zuckerberg	Salary: 30395.94	Employees: 10
ID: 24	Name: Shlomo Noyce	Boss: Taylor Zuckerberg	Salary: 29222.68	Employees: 10
ID: 35	Name: Kate Filo	Boss: Taylor Zuckerberg	Salary: 25677.13	Employees: 10
<b>Team members:</b>				
ID: 32	Name: Max Noyce	Boss: Kate Hewlett	Salary: 20675.38	Language: Java
ID: 40	Name: Lucy Jobs	Boss: Max Ballmer	Salary: 19595.35	Language: C++
ID: 16	Name: Imen Moore	Boss: Shlomo Noyce	Salary: 19509.67	Language: Ruby

# איך מייצרים דו"ח?

- שימוש ב-instanceof במתודת יצירת דו"ח
- שימוש ב-toString (או מתודה ייעודית)
  - תלוי במספר מצומצם של פורמטים/דו"חות?
- שימוש במחלקה ייעודית לכל דו"ח
  - תלוי בכך שאין שינויים רבים במחלקות

# toString()

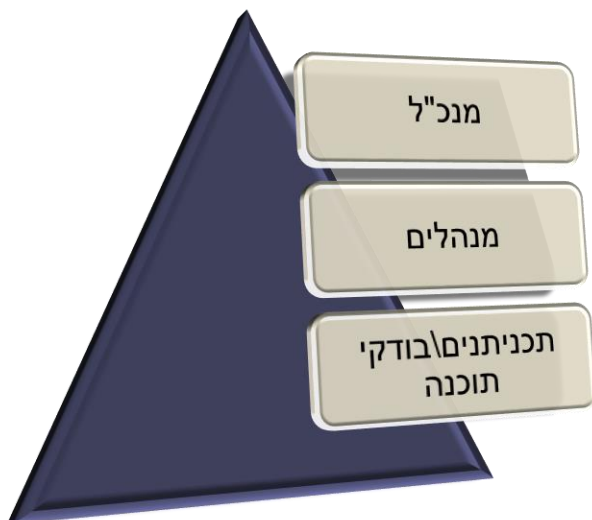
```
public abstract class AbstractEmployee implements Employee {
...
    @Override
    public String toString() {
        StringBuilder str = new StringBuilder();
        str.append("ID: ").append(id);
        str.append("\tName: ").append(name);
        str.append("\tBoss: ");
        if (getBoss() != null)
            str.append(getBoss().getName());
        else
            str.append("None");
        str.append("\tSalary: ");
        str.append(String.format("%.2f", getSalary()));

        return str.toString();
    }
}
```

# toString()

```
public class QATester extends TeamMember {  
    ...  
    @Override  
    public String toString() {  
        return super.toString() + "\tBugs found: " + getBugsFound();  
    }  
}
```





## עוד דרישות:

- כתבו תכנית המייצרת אובייקטים של עובדים עם נתונים אקראיים ושומרת אותם בשלוש רמות היררכיות לפי הפירוט הבא:
  - בראש ההיררכיה נמצא המנכ"ל שהינו מנהל
  - מתחתיו בהיררכיה יש 5 מנהלים
  - מתחת לכל מנהל מצויים בהיררכיה 10 תכניתנים או בודקי תוכנה (בהסתברות שווה).
- לאחר מכן, התוכנית תדפיס את פרטי 3 העובדים עם המשכורת הגבוהה ביותר בכל רמה היררכית.

# Sorting by salary

- נגדיר השוואה מתאימה:

```
public class SalaryComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee o1, Employee o2) {
        return Double.compare(o2.getSalary(), o1.getSalary());
    }
}
```

מיון בסדר הפוך – מהגדול לקטן

- כעת נוכל לייצר את הדו"ח

```
public static void printTopPaid(List<Employee> employees) {
    Collections.sort(employees, new SalaryComparator());
    for(int i=0; i<3; ++i)
        System.out.println(employees.get(i));
}
```

## ראינו היום

- תכנון היררכית מחלקות וירושה
- קצת enums
- "חלוקת אחריות" על פעולה בין מחלקות
- מתודות חשובות מ-Object: toString, equals, hashCode
- עוד דוגמאות לשימוש באוספים גנריים ומיון רשימות

# THE END

הקוד נמצא במלואו באתר הקורס