חריגים

- נממש שירות המחשב ממוצע הרמוני על אוסף של מספרים.

$$H = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}$$

```
public static double harmonicMean(Collection<Integer> numbers){
     if (numbers.isEmpty()){
          return 0;
     }
     double denominator = 0;
     for (int i : numbers){
          denominator += 1.0/i;
     }
     return numbers.size()/ denominator;
}
```

שאלה: ממוצע הרמוני מוגדר רק על מספרים חיוביים. מה נעשה אם נקבל מספר אי-חיובי ברשימה?

חריגים

- אופציה ראשונה:
  - נקבל החלטה בתוך השירות, למשל:
    - נתעלם מהמספרים האי-חיוביים ונחשב ממוצע הרמוני על שאר המספרים.
    - נחזיר 0 או מספר ברירת מחדל אחר
  - חסרונות – המשתמש לא ידע שמשהו לא תקין, אם היה יודע, אולי היה מעדיף דרך אחרת לטיפול.
- אופציה שניה:
  - שימוש בחריגים.

חריגים

```
public static double harmonicMean(Collection<Integer> numbers) throws Exception{
     if (numbers.isEmpty()){
          return 0;
     }
     double denominator = 0;
     for (int i : numbers){
          if (i <= 0){
               throw new Exception("wrong value in list: " + i);
          }
          denominator += 1.0/i;
     }
     return numbers.size()/denominator;
}
```

עלינו לייצר אובייקט חדש מטיפוס Exception ולהשתמש במילה השמורה throw בשביל לזרוק את השגיאה

מצהירים על שגיאה שנזרקת בשירות

חריגים

- נוסיף שירות נוסף – השירות מקבל מפה: משם קובץ לאוסף המספרים שהוא מכיל, ומדפיס ממוצע הרמוני עבור כל קובץ.

```
public static void printMeansByFiles(Map<String, Collection<Integer>> numbers) {
     for (Map.Entry<String, Collection<Integer>> mapEntry: numbers.entrySet()){
          double hMeanForFile = harmonicMean(mapEntry.getValue());
          System.out.println("for file: " + mapEntry.getKey() + "
                                          hMean is: " + hMeanForFile);
     }
}
```

Exception

בקוד הזה יש שגיאת
קומפילציה בגלל שגיאה
שלא הצהרנו עליה אך גם לא
טיפלנו בה

חריגים

- אפשרות ראשונה: לא נטפל בחריג, ורק נצהיר עליו
- במקרה הזה, מי שיצטרך להתמודד עם הטיפול בחריג הוא השירות שיקרא ל printMeansByFiles.

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
                                                                    throws Exception{
     for (Map.Entry<String, Collection<Integer>> mapEntry: filesInfo.entrySet()){
          double hMeanForFile = harmonicMean(mapEntry.getValue());
          System.out.println("for file: " + mapEntry.getKey() + "
                                          hMean is: " + hMeanForFile);
     }
}
```

חריגים

- אפשרות שניה: נטפל בחריג!

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo) {
     for (Map.Entry<String, Collection<Integer>> mapEntry: filesInfo.entrySet()){
          try{
               double hMeanForFile = harmonicMean(mapEntry.getValue());
               System.out.println("for file: " + mapEntry.getKey() + " hMean is: "
                                                  + hMeanForFile);
          }
          catch (Exception e){
               System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
          }
     }
}
```

חריגים

- איך זה עובד?

```
public static void main(String[] args){
          Map<String, Collection<Integer>> files = new LinkedHashMap<>();
          files.put("file1", Arrays.asList(1, 2, 3));
          files.put("file2", Arrays.asList(1,2,-4));
          files.put("file3", Arrays.asList(15,17,30));
          printMeansByFiles(files);
}
```

- תוכנית זו מייצרת את הפלט:

```
for file: file1 hMean is: 1.6363636363636365
cannot calculate hMean for file file2
for file: file3 hMean is: 18.88888888888889
```

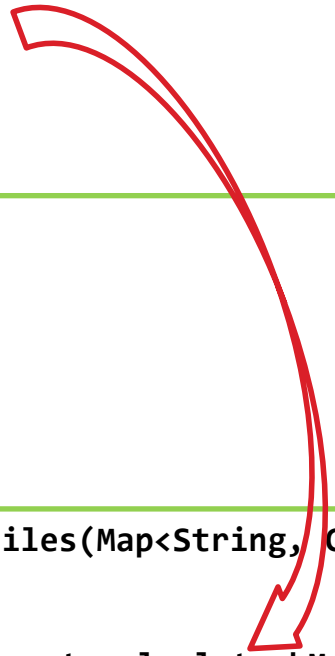
חריגים

- ובכל זאת יש בעיה – אנחנו מטפלים בכל שגיאה אפשרית שיכולה להיזרק מתוך harmonicMean, ועל הדרך יכולים להתעלם משגיאות שמעידות על באג אפשרי.
- במימוש שלנו הנחנו הנחה סמויה לגבי המפה, למרות שאין לנו דרך לדעת כיצד היא נוצרה (נניח שאין חוזה לשירות).
- מה יקרה במקרה הבא?

```
public static void main(String[] args){
        Map<String, Collection<Integer>> files = new LinkedHashMap<>();
        files.put("file1", null);
        files.put("file2", Arrays.asList(1,2,-4));
        files.put("file3", Arrays.asList(15,17,30));
        printMeansByFiles(files);
}
```

חריגים

```
public static double harmonicMean(Collection<Integer> numbers) throws Exception{
     if (numbers.isEmpty())
     …
}
```



NullPointerException

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
     …
     catch (Exception e){
          System.out.println("cannot calculate hMean for file " +  mapEntry.getKey());
     }
     …
}
```

חריגים

- מה נרצה לעשות במידה והמפה שלי מכילה null?
  - יכול להיות שנרצה להתייחס לזה כמו לרשימה ריקה (שזה למעשה הטיפול שקיים כרגע בקוד).
  - יכול להיות שנרצה להדפיס הודעה למשתמש: המפה מכילה null, אולי קרתה שגיאה בטעינת הקובץ?
  - יכול להיות שנרצה לזרוק את השגיאה ולהטיל את הטיפול על מי שמשתמש ב `printMeansByFiles`
- אם נרצה להתייחס למקרה של מפה המכילה null באופן שונה ממפה המכילה מספר לא חיובי, עלינו לדעת להבדיל בין החריגים.
  - הצעה: נוסיף בלוק except עבור NullPointerException
    - ומה אם יש עוד שגיאות שיכולות להיזרק?

יצירת טיפוס חריג חדש

ירושה מ Exception

```
class HMeanException extends Exception{
     public HMeanException(String message) {
          super("Harmonic Mean calculation error! " + message);
     }
}
```

קריאה לבנאי של מחלקת האב – קריאה זו תמיד תהיה הפקודה הראשונה של הבנאי

שימוש בטיפוס החריג החדש

```
public static double harmonicMean(Collection<Integer> numbers) throws HMeanException {
     if (numbers.isEmpty()){
          return 0;
     }
     double denominator = 0;
     for (int i : numbers){
           if (i <= 0){
                     throw new HMeanException("wrong value in list: " + i);
           }
          denominator+ = 1.0/i;
     }
     return numbers.size()/denominator;
}
```

שימוש בטיפוס החריג החדש

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
   …
   catch (HMeanException e){
     System.out.println("cannot calculate hMean for file " +  mapEntry.getKey());
   }
   …
}
```

הבלוק הזה יטפל רק
בשגיאה שזרקנו מתוך
harmonicMean, חריגים
אחרים יזרקו הלאה.

שימוש בשגיאות – פורמט הודעת השגיאה

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
   …
   catch (HMeanException e){
     System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
        e.printStackTrace();
   }
}
public static void main(String[] args){
   Map<String, Collection<Integer>> files = new LinkedHashMap<>();
   files.put("file2", Arrays.asList(1,2,-4));
   files.put("file3", Arrays.asList(15,17,30));
   printMeansByFiles(files);
}
```

- עבור תוכנית זו נקבל את הפלט:

```
for file: file1 hMean is: 1.6363636363636365
cannot calculate hMean for file file2
HMeanException: Harmonic Mean calculation error! wrong value in list: -4
        at Tmp.harmonicMean(Tmp.java:22)
        at Tmp.printMeansByFiles(Tmp.java:36)
        at Tmp.main(Tmp.java:52)
for file: file3 hMean is: 18.88888888888889
```

שימוש בשגיאות

- הדפסת פורמט שגיאה מצומצם יותר:

```
public static void printMeansByFiles(Map<String, Collection<Integer>> filesInfo)
   …
   catch (HMeanException e){
        System.out.println("cannot calculate hMean for file " + mapEntry.getKey());
        System.out.println(e.getMessage())
   }
}
```

- פלט התוכנית יהיה:

```
cannot calculate hMean for file file2
Harmonic Mean calculation error! wrong value in list: -4
for file: file3 hMean is: 18.88888888888889
```

```
class HMeanException extends Exception{
     public HMeanException(String message) {
          super("Harmonic Mean calculation error! " + message);
     }
```

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 וירושה
  - אוספים
- במסגרת התרגיל נכתוב תכנית לחישוב שכר בחברת הייטק המורכבת ממספר סוגים של עובדים.

עצבו מחלקות לייצוג עובדים בחברה על פי המפרט הבא:

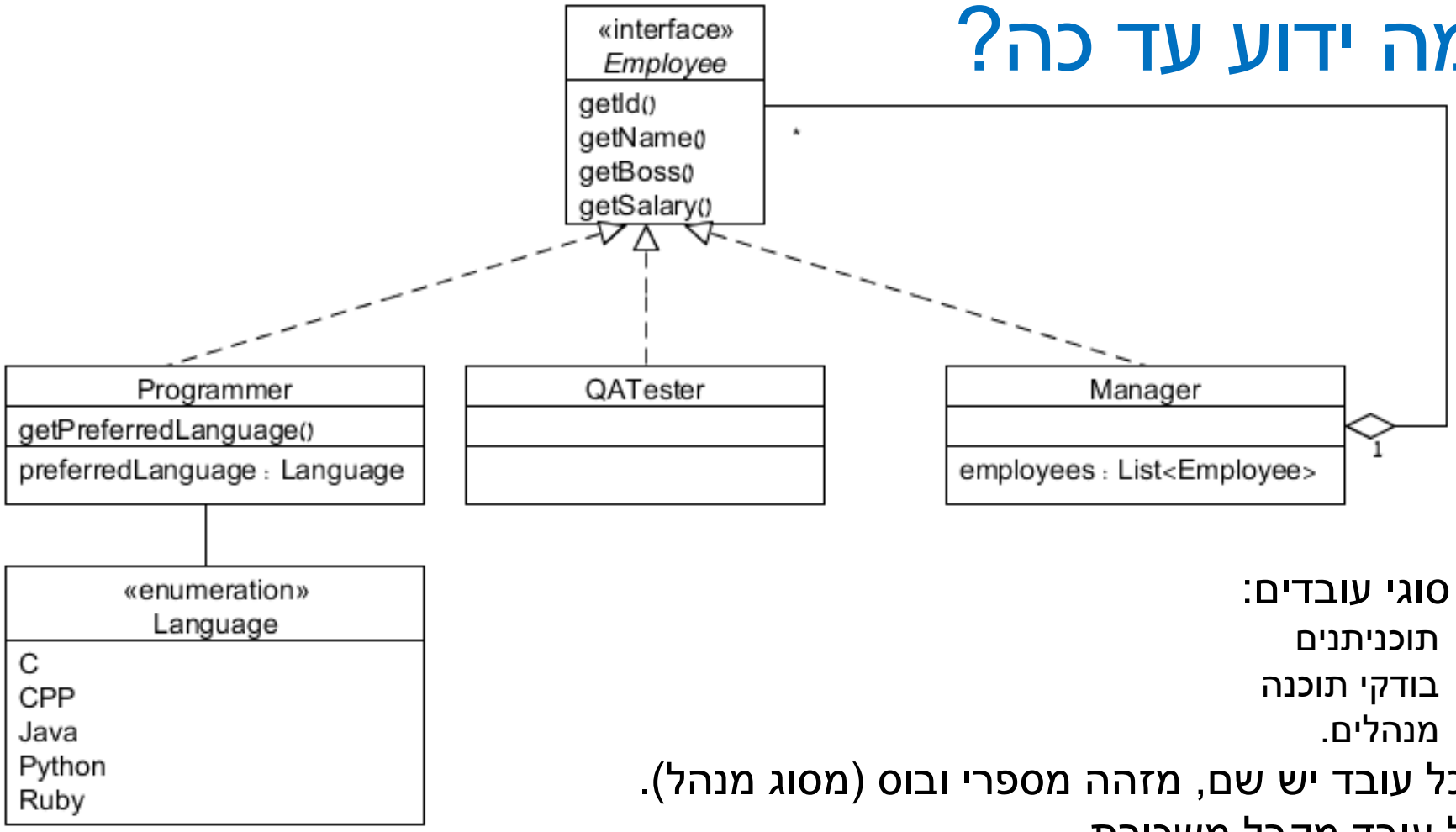
- בחברת הייטק מצליחה ישנם 3 סוגי עובדים:
  - תוכניתנים
  - בודקי תוכנה
  - מנהלים.
- לכל עובד יש:
  - שם
  - מזהה מספרי
  - בוס (מסוג מנהל).
- כל עובד מקבל משכורת.
- לכל מנהל יש רשימה של עובדים אותם הוא מנהל.
- לכל תוכניתן יש שפת תכנות מועדפת (מתוך רשימה אפשרית)

המשך המפרט:

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

נתחיל?

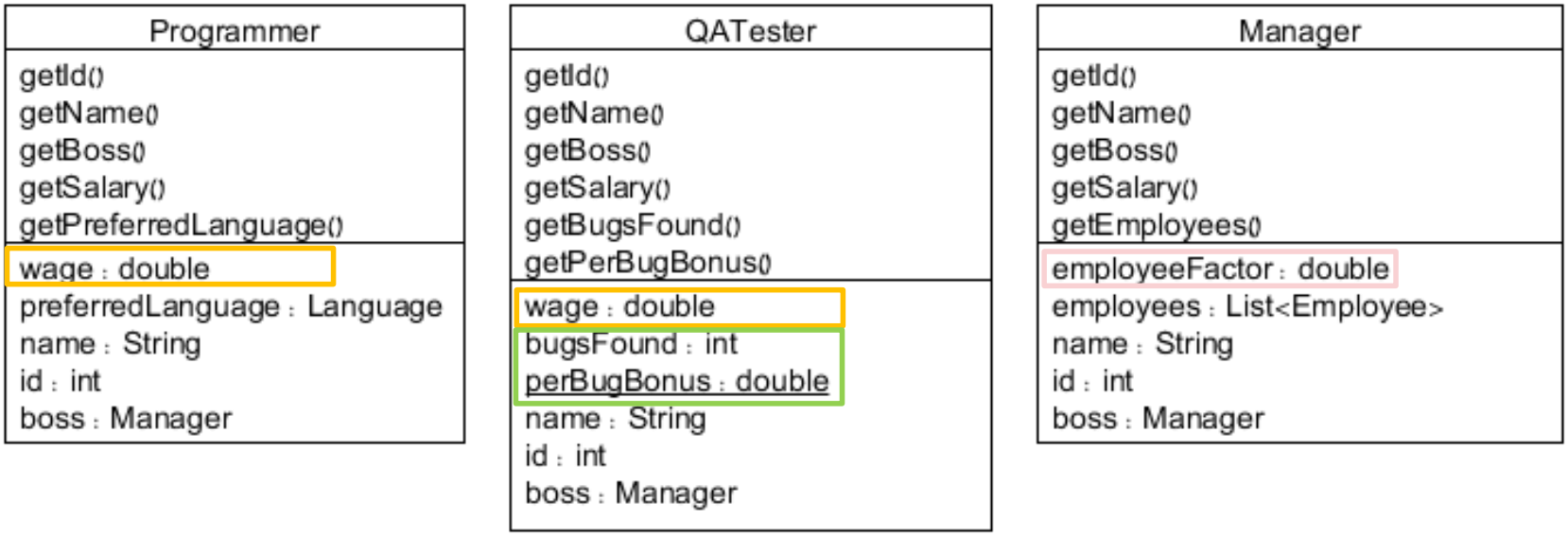
מה ידוע עד כה?



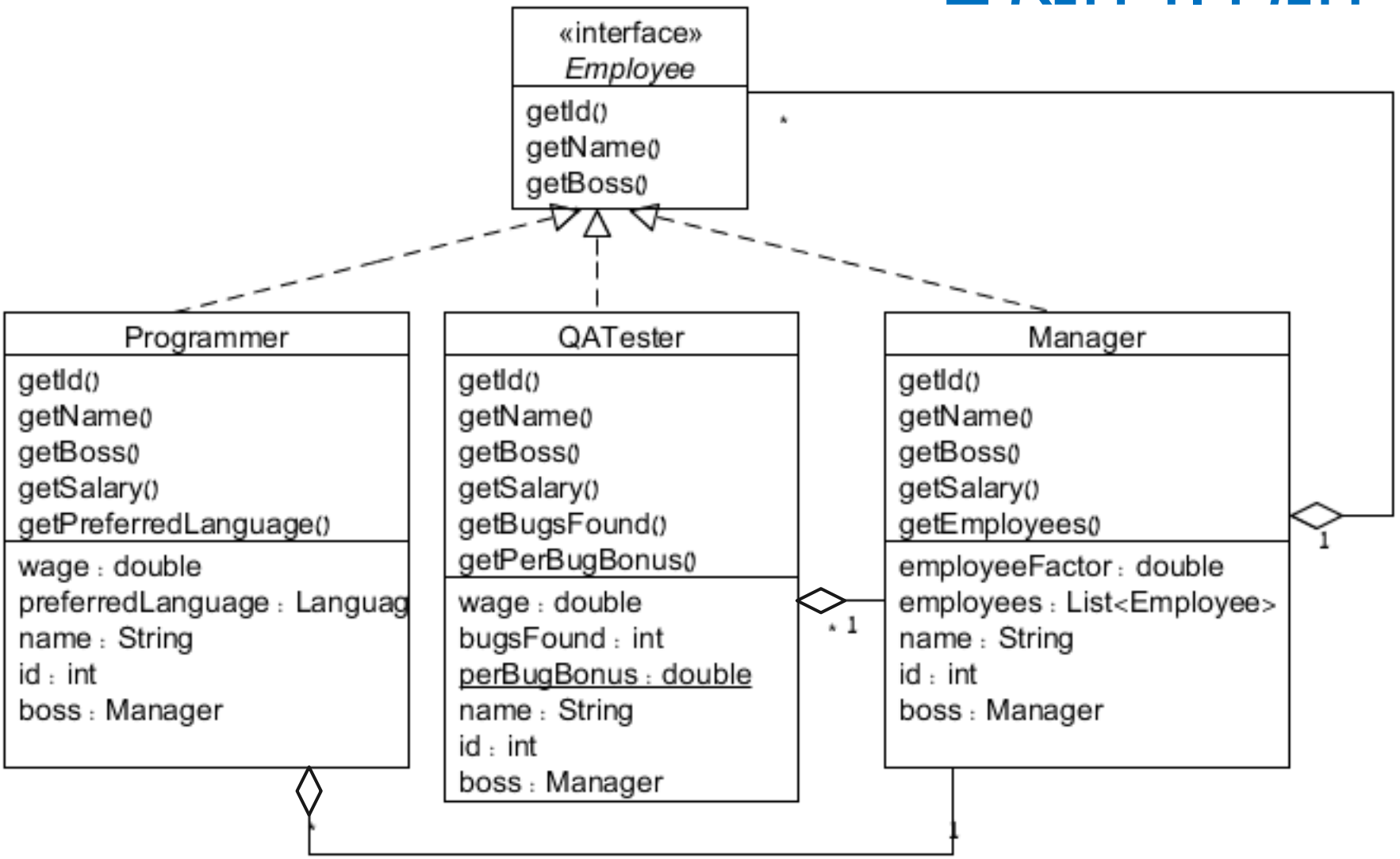
- 3 סוגי עובדים:
  - תוכניתנים
  - בודקי תוכנה
  - מנהלים.
- לכל עובד יש שם, מזהה מספרי ובוס (מסוג מנהל).
- כל עובד מקבל משכורת.
- לכל מנהל יש רשימה של עובדים אותם הוא מנהל.
- לכל תוכניתן יש שפת תכנות מועדפת (מתוך רשימה אפשרית)

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.

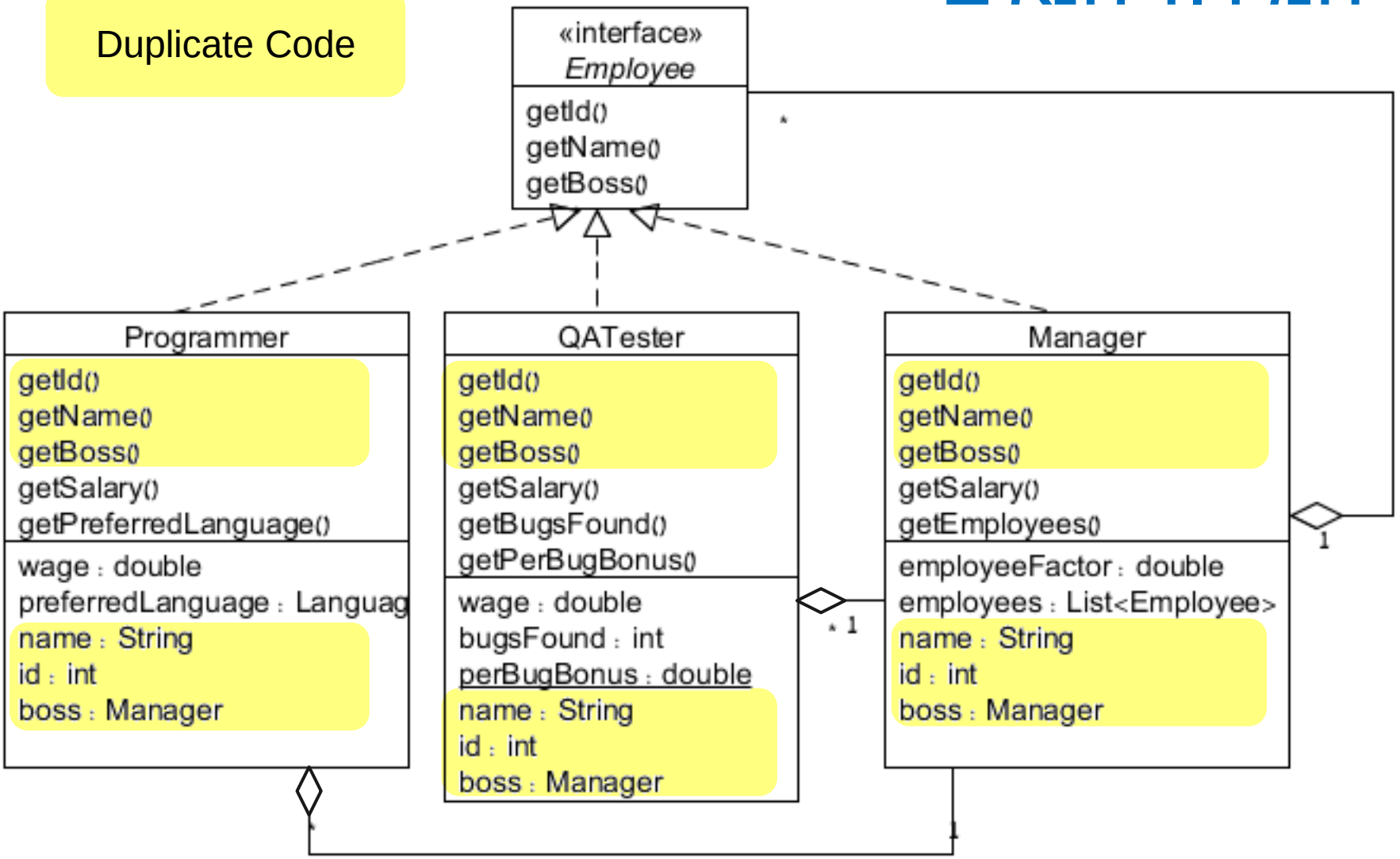


המידול הנאיבי



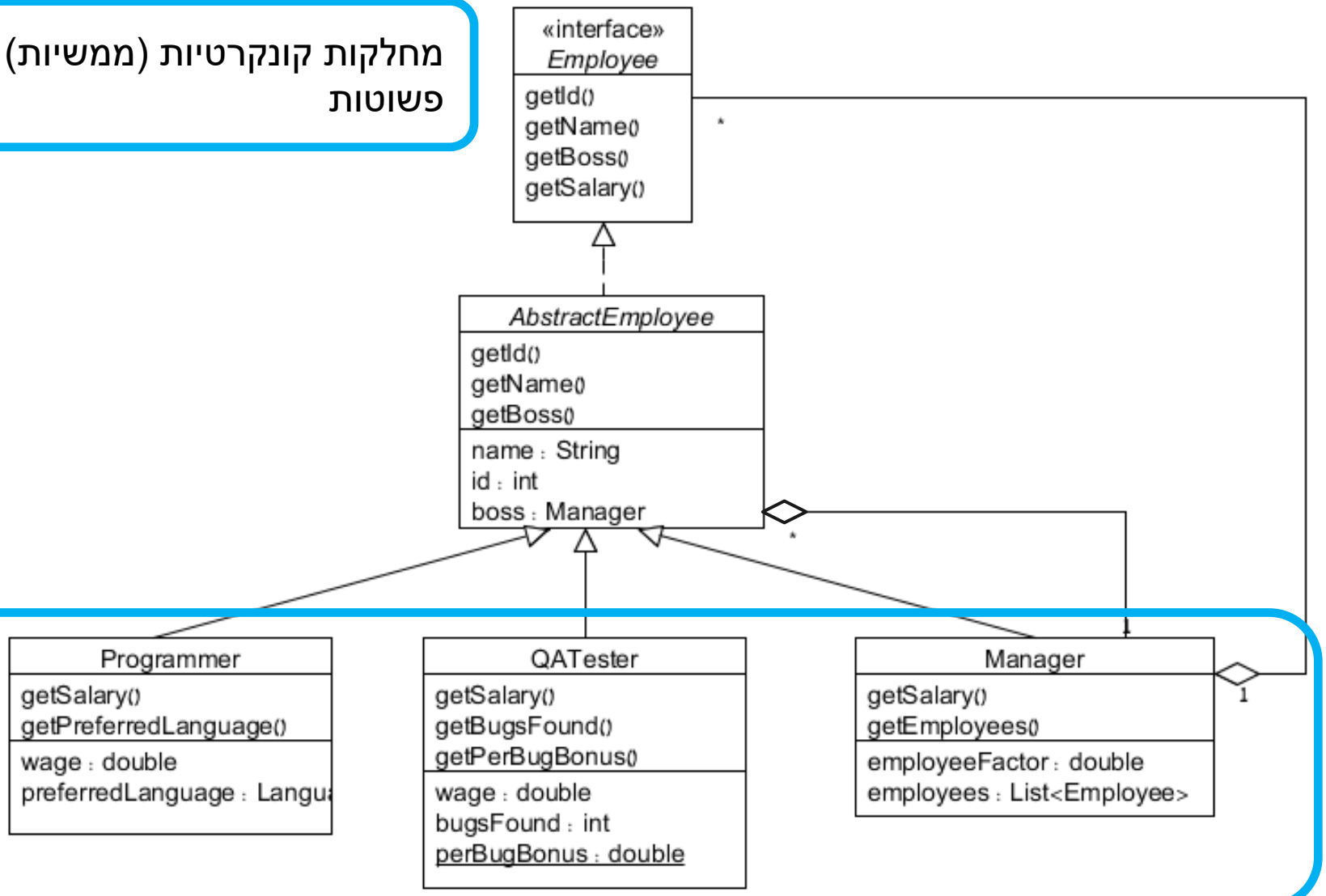
המידול הנאיבי

Duplicate Code



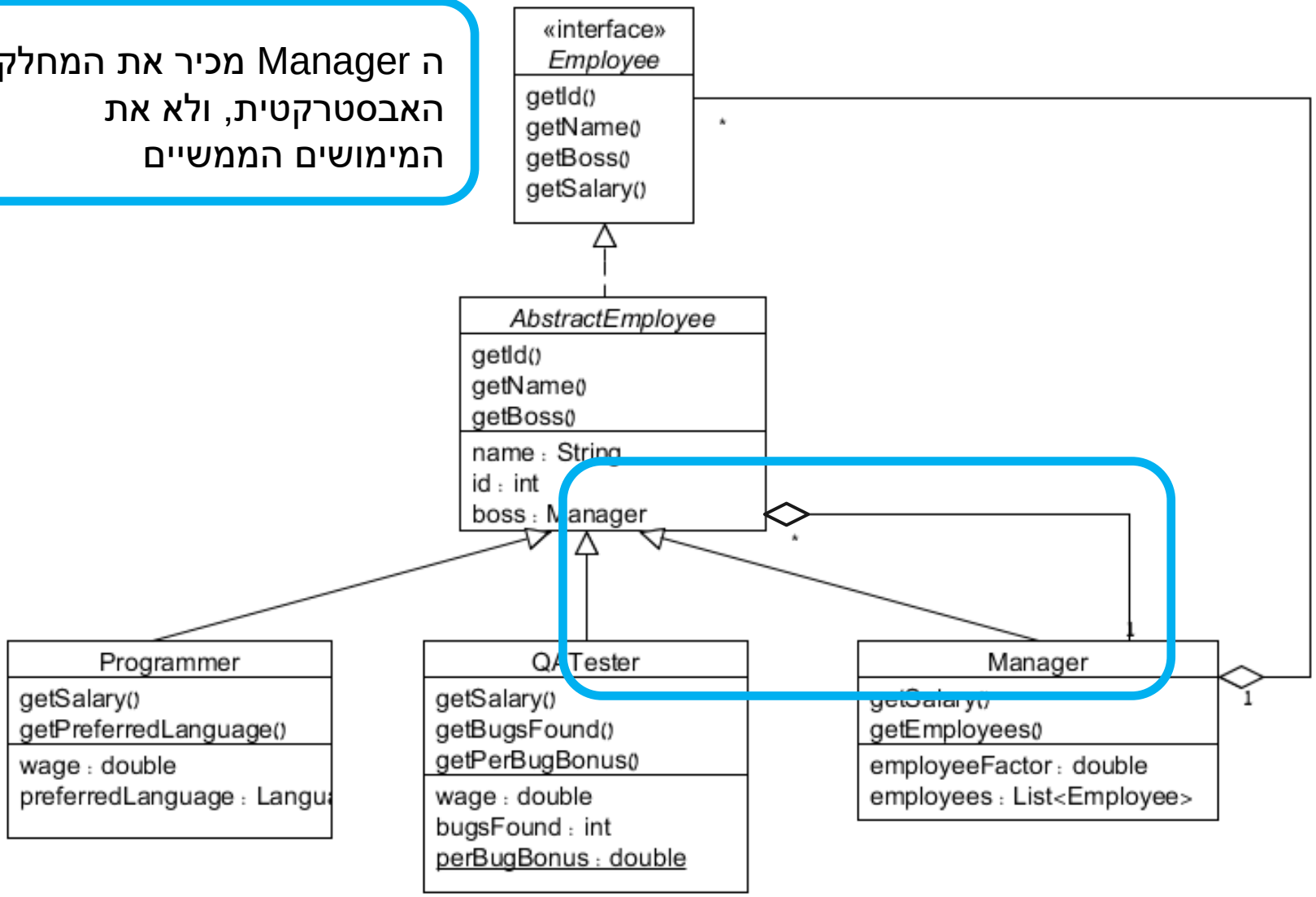
שלב 1 – עובד אבסטרקטי

מחלקות קונקרטיות (ממשיות) פשוטות



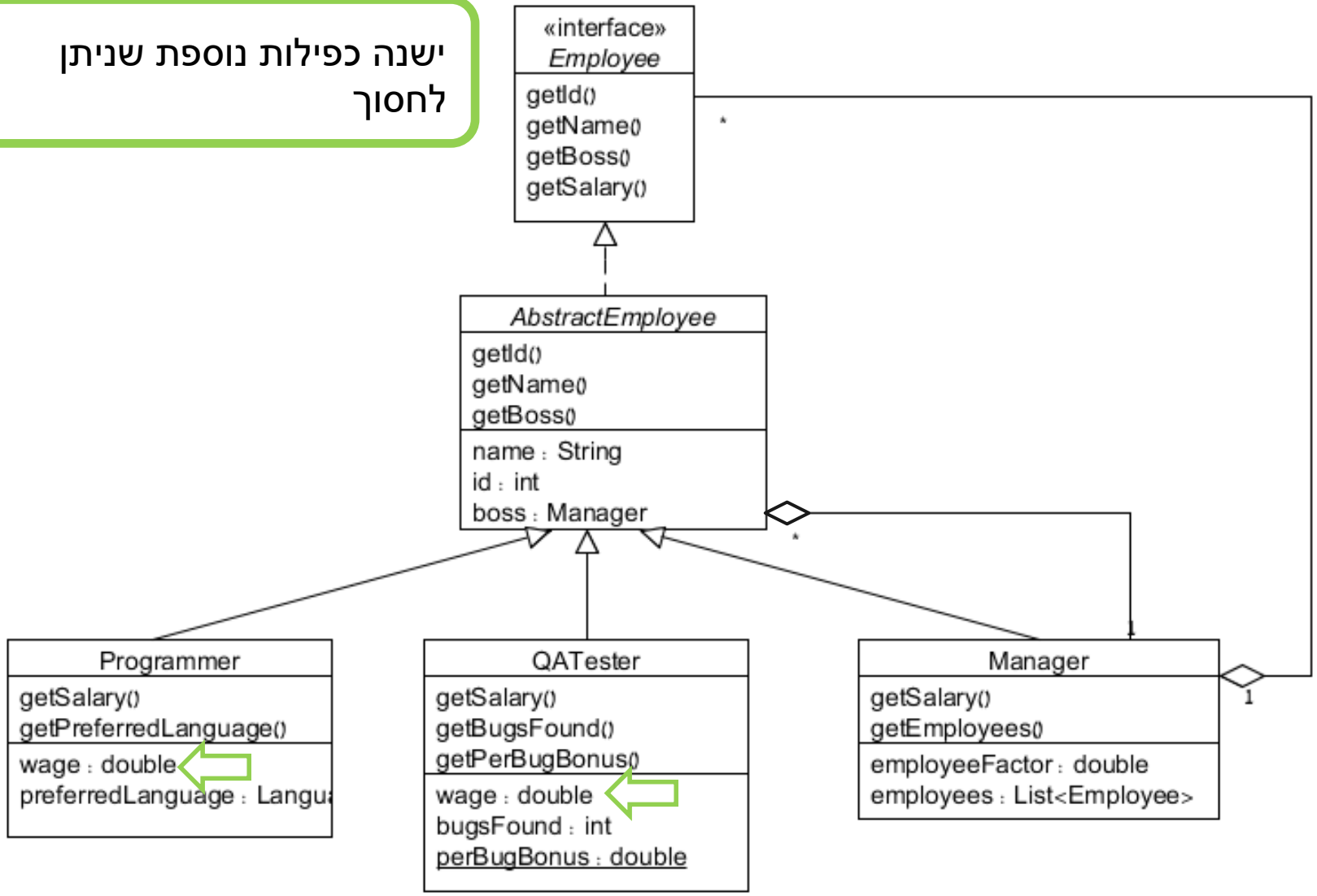
שלב 1 – עובד אבסטרקטי

ה Manager מכיר את המחלקה האבסטרקטית, ולא את המימושים הממשיים

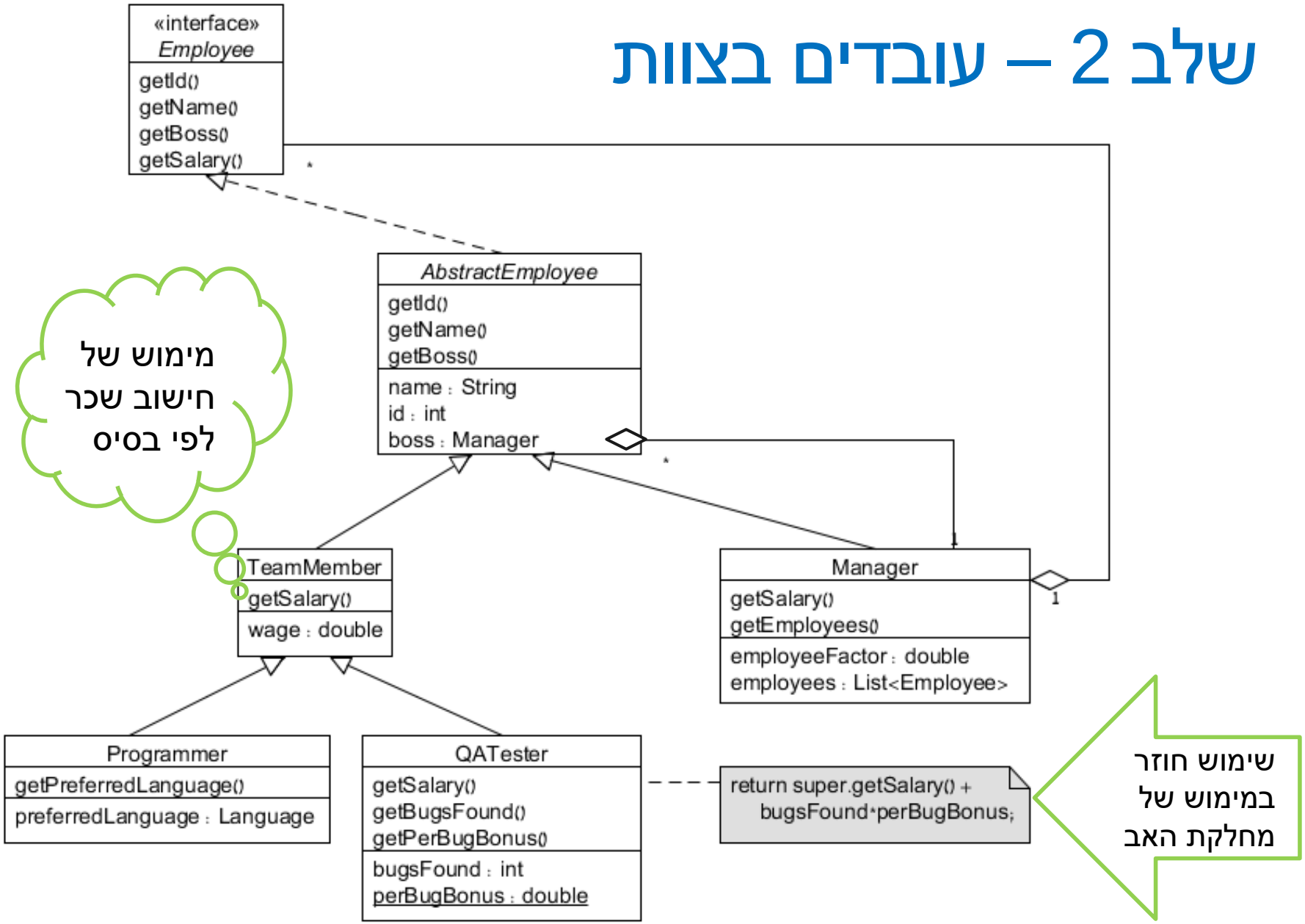


שלב 1 – עובד אבסטרקטי

ישנה כפילות נוספת שניתן לחסוך

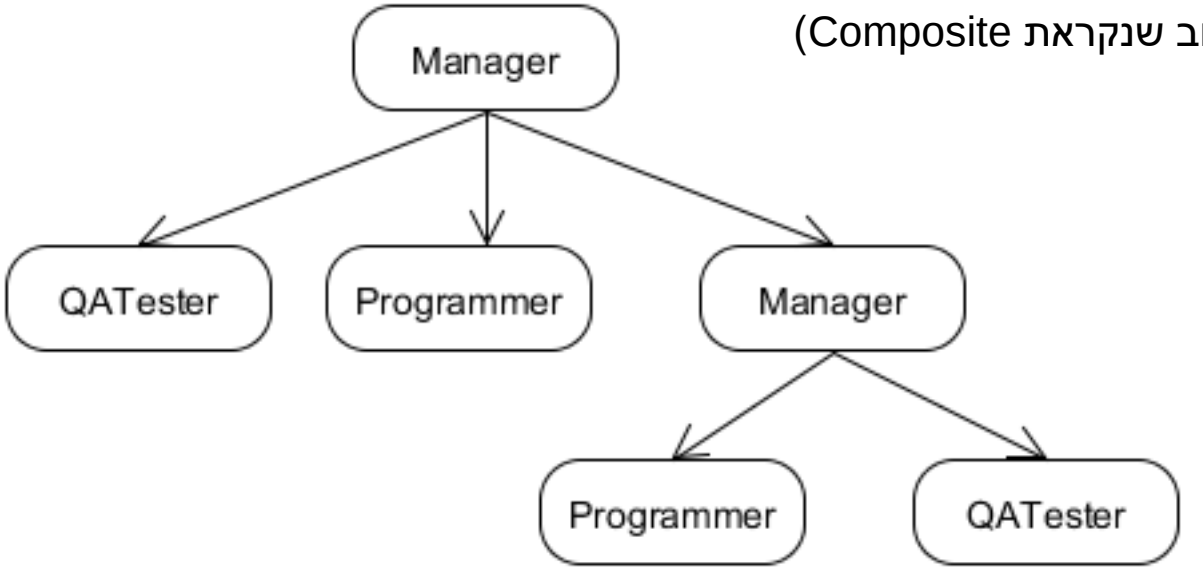


שלב 2 – עובדים בצוות

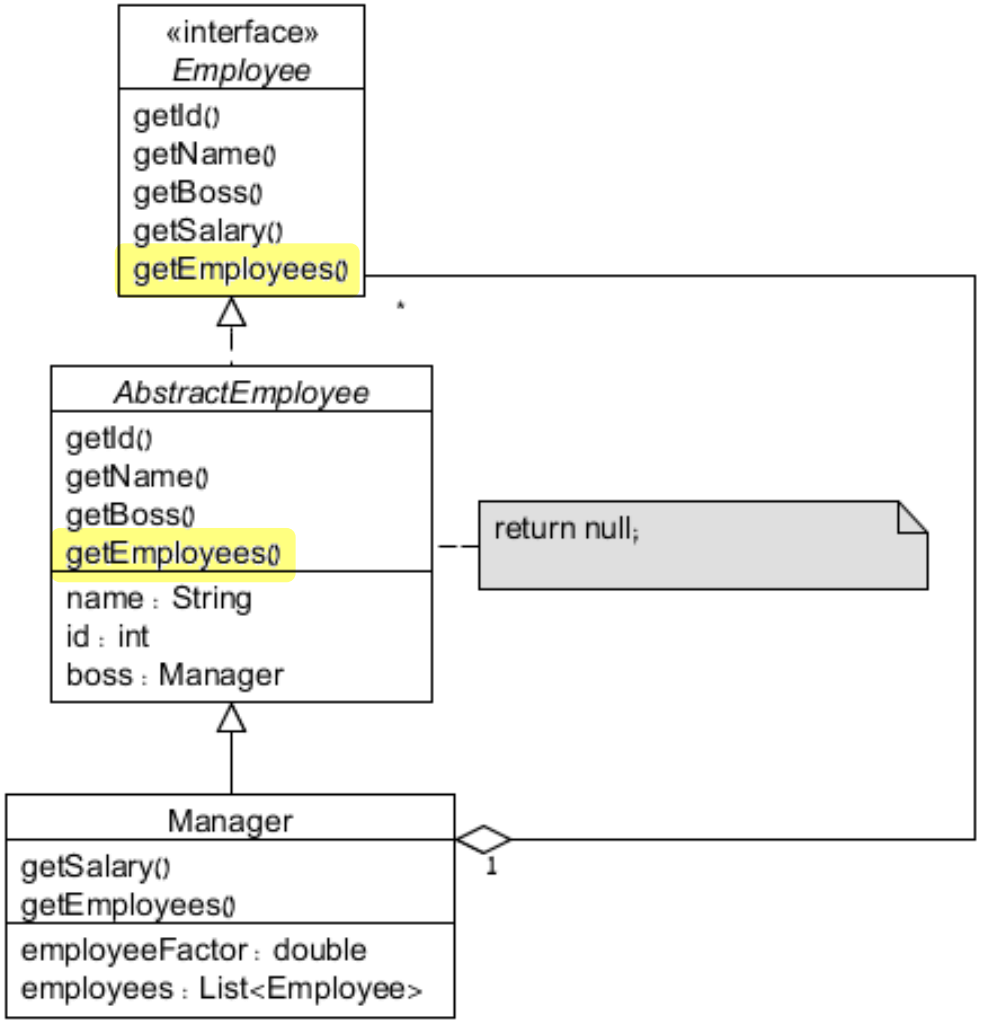


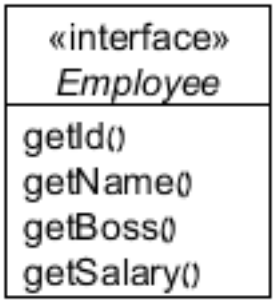
שלב 3 – plan ahead? (אופציונאלי)

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



שלב 3 – plan ahead? (אופציונאלי)





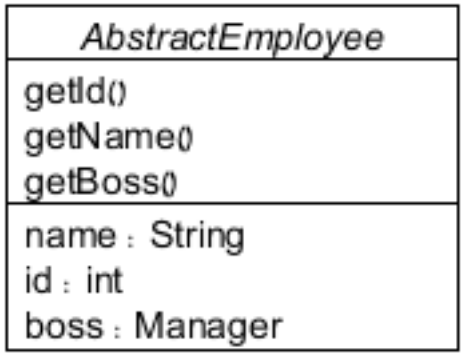
מה הלאה?

- 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();
}
```

עוד קוד

```
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;
    }
}
```



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;
    }
}
```

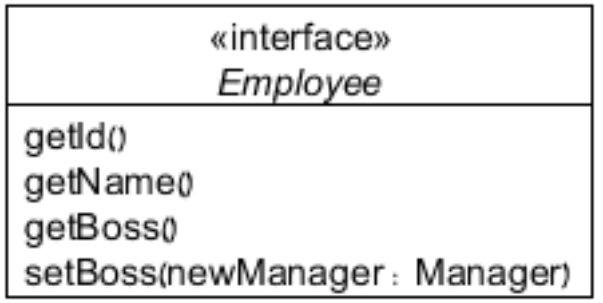
Enumerated types - usage

```
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

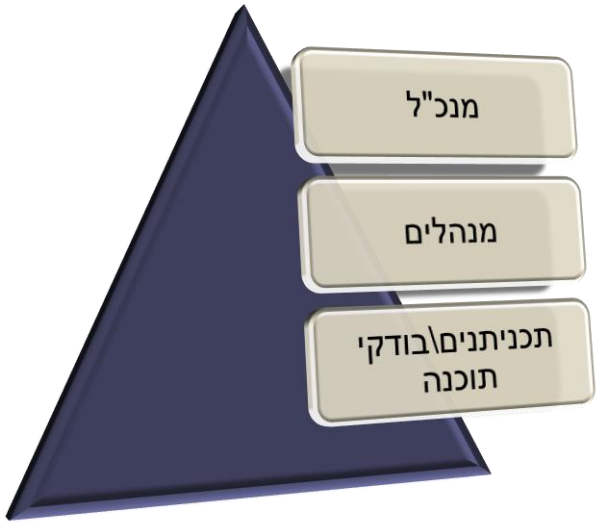
```
public class TeamMember extends AbstractEmployee {

    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 + בונוס

```
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()   { retrun 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:

CEO:

ID: 1	Name: Taylor Zuckerberg	Boss: None	Salary: 49740.43	Employees: 5

Managers:

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

Team members:

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

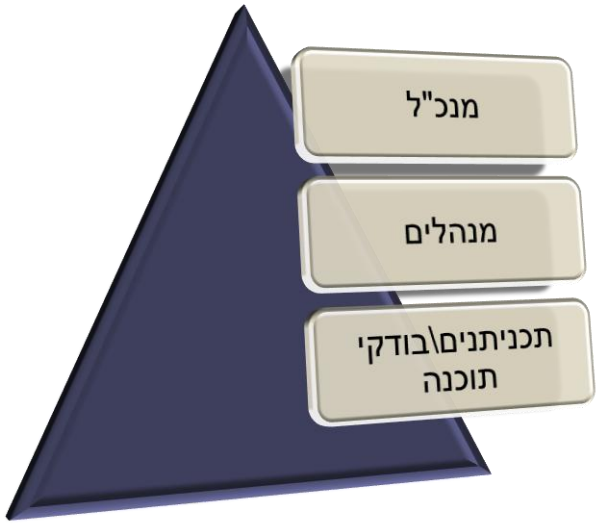
toString()

```
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