

מערכת תשלומים (תרגיל 9, שאלה 1): פתרון לדוגמא.

ראשית, נשים לב שייטכנו מספר מימושים אפשריים.

במימוש טריוויאלי נגדיר היררכית אובייקטים ששורשה במחלקה Employee, וכל אובייקט נטפל במשכורת עפ"י הכללים האמורים. על מנת לבצע את העדכונים הדרשים בכל "שבוע" נעבור על רשימת האובייקטים ובעזרת האופרטור instanceof נברר מיהו האובייקט הקונקרטי היורש מ-Employee ונעדכן משכורתו בהתאם.

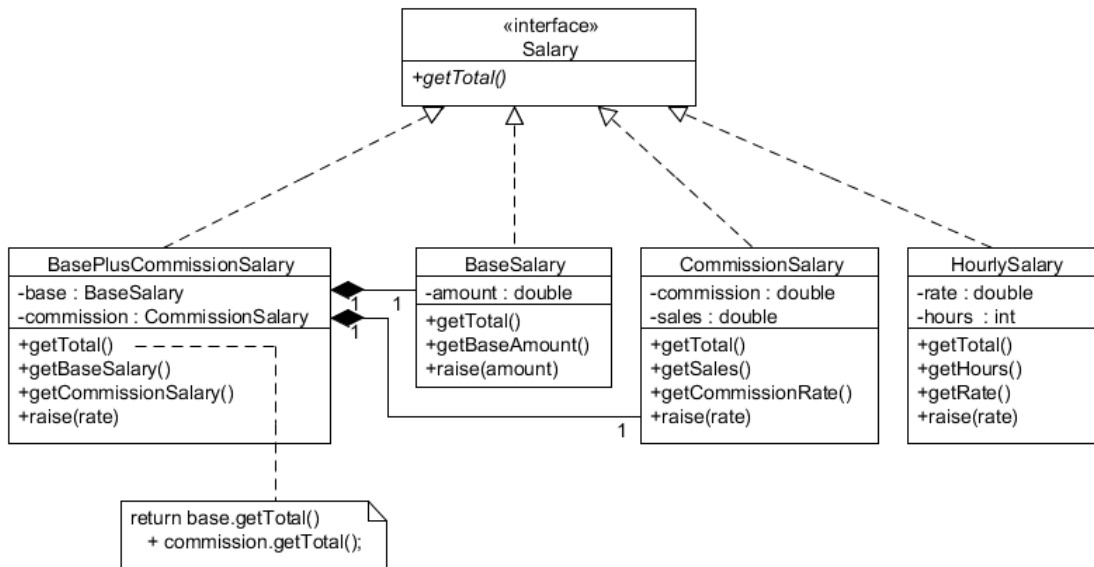
לפתרון זה מספר חסרונות:

1. הוא מסתמך על מנגנון שפה ספציפי (instanceof)
2. קשה לתחזק אותו: בקוד יהיה לנו משפט תנאי (if) הבודק את כל אחד מסוגי ה-Employee המוגדרים. לכן, אם נרצה להוסיף חישוב שכר חדש המבוסס על תמהיל שונה של רכיבי שכר (לדוגמא BasePlusHourlyEmployee) נאלץ להוסיף קוד נוסף למשפט ה-if.

נתאר בפירוט פתרון אחר.

ראשית, נשים לב כי אין שוני רב בין מחלקות העובדים (השוני העיקרי הוא במתודה toString). עיקר השוני הנדרש הוא בחישוב המשכורת. לכן, נתאר את המשכורת כיישות נפרדת. נגדיר מנשק Salary המאפשר חישוב משכורת עפ"י רכיבי השכר השונים. רכיבי השכר הבסיסיים יממשו את המנשק עפ"י הכללים השונים. את רכיבי השכר מורכבים (BasePlusCommission) נאגד ע"י תבנית העיצוב Composite שהכרנו בתרגיל 8. לשם פשטות נממש Composite המורכב משני רכיבי השכר הספציפיים (ניתן היה להכיל Collection כללי של אובייקטים היורשים מ-Salary).

דיאגרמת המחלקות הבאה מתארת את היררכית מחלקות ה-Salary:



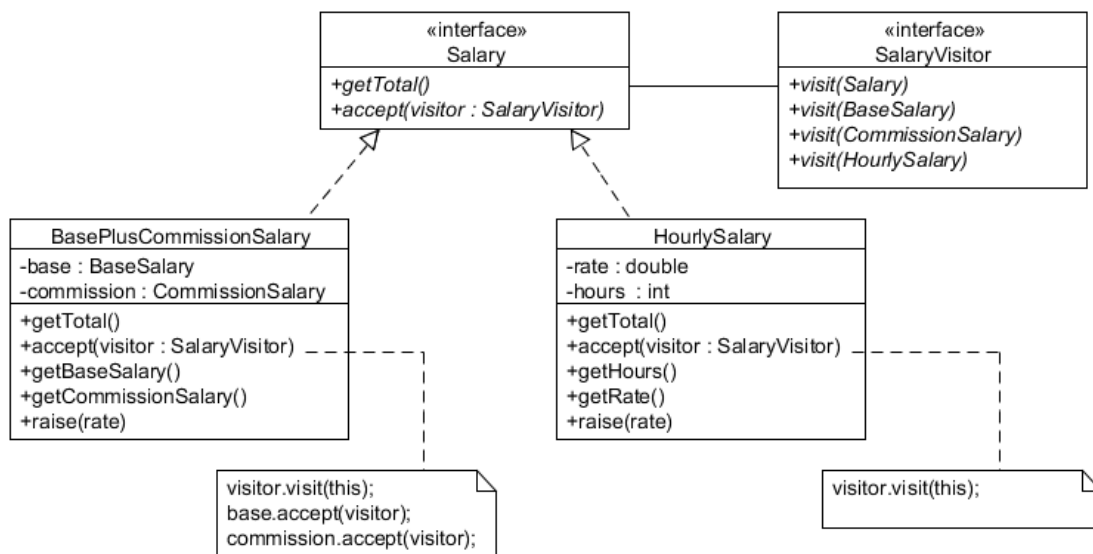
כעת נרצה לטפל בעדכוני משכורות. נשים לב ש:

1. רכיבי השכר השונים מעודכנים בצורה שונה מזמן לזמן.
2. העדכון מתבצע עבור רכיבי שכר בסיסיים.

נרצה דרך לעבור על המשכורות ולטפל ברכיבי השכר הבסיסיים השונים. חשוב: בהינתן מבנה ה- Composite של רכיבי שכר, אנו מטפלים באובייקטים שאינם בהכרח בעלי מבנה "שטוח", לדוגמא: אובייקטים מורכבים כגון BasePlusCommissionSalary נסמכים על מופעים של אובייקטים נוספים מסוג Salary ולכן כללית, לא נרצה להניח הנחות על מבנה הנתונים.

כבר ראינו בתרגיל 8 דרך לעבור על מבנה דינמי (במקרה שלנו המבנה מוגדר באופן שונה לכל עובד), המכיל רכיבים הבסיסיים המשתנים בתדירות נמוכה (אם בכלל). פתרנו זאת בעזרת תבנית העיצוב Visitor. נגדיר, אם כן את המנשק SalaryVisitor, ונוסיף למנשק Salary את המתודה accept המתאימה.

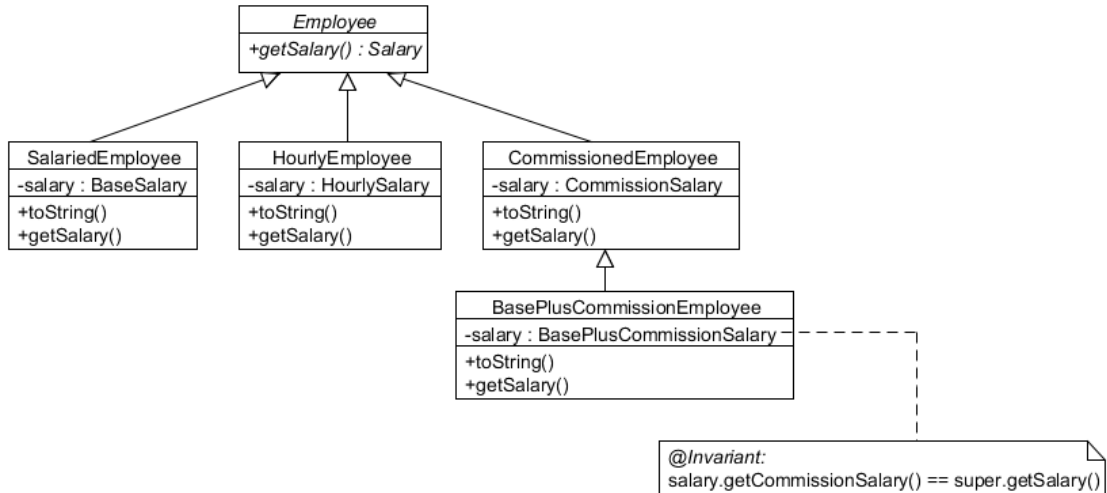
תזכורת לתבנית העיצוב Visitor (עבור חלק מהמחלקות):



עפ"י הנתון בתרגיל, בד"כ נרצה לעדכן רק רכיב שכר יחיד. לשם פשטות, נגדיר את המחלקה האבסטרקטית SalaryVisitorAdapter המספקת מימוש ריק לכל המתודות של המנשק SalaryVisitor:

1. המחלקה אבסטרקטית כי איננה מבצעת דבר ולא סביר שנרצה לעבור על רכיבי השכר הבסיסיים בלי לבצע דבר.
2. המחלקה מאפשרת לנו לממש תיקון לרכיב שכר יחיד בצורה פשוטה וקריאה (ראו בהמשך).

נותר לתאר את המחלקות המטפלות בעובדים. מחלקות אלו הן מחלקות פשוטות הנבדלות זו מזו בעיקר במתודה toString שלהן. בנוסף, כל המחלקות מחזיקות אובייקט היורש מ-Salary ומטפל בחישוב המשכורת. בחרנו לממש את ההיררכיה הבאה המאפשרת חסכון בקוד במימוש toString עפ"י הנדרש:



כעת, בהינתן בסיס נתונים של עובדים, ניתן לתקן רכיב שכר בצורה פשוטה. לדוגמא, כדי להוסיף 10% לשכר של שעת עבודה (השינוי הנדרש לאחר השבוע הראשון) נוכל להשתמש בקטע הקוד הבא:

```

for (Employee e : employees) {
    e.getSalary().accept(new SalaryVisitorAdapter() {
        @Override
        public void visit(HourlySalary salary) {
            salary.raise(salary.getRate() * 0.1);
        }
    });
}
  
```

שימו לב ש-SalaryVisitorAdapter מאפשר לכתוב קוד עבור הרכיב אותו נדרש לשנות בלבד ומשחרר אותנו מהצורך לממש את כל מתודות האחרות של המנשק ShapeVisitor.

לפתרון שהצענו מספר ייתרונות:

1. איננו נסמכים על מנגנון השפה "instanceof" או מנגנונים אחרים המבררים את סוג (type) האובייקט בזמן הריצה.
2. המנגנון מאפשר הוספת עובדים אשר המשכורות שלהן משלבות רכיבי שכר שונים בצורה קלה יחסית.
3. ניתן להחזיק את רשימת העובדים בבסיס נתונים יחיד מבלי לאבד את היכולת לבצע עדכוני שכר פרטניים (כלומר כאלו הרלוונטיים רק לחלק מהעובדים).

החיסרון העיקרי של הפתרון המוצע הוא שהמערכת הנוצרת היא מעט מורכבת ומי שאינו מכיר את תבניות העיצוב בהן השתמשנו עלול להתקשות בהבנתה.