CS1083 Week 3: Review

Method Decomposition, Aggregation, Inheritence

David Bremner

2018-01-17

Method decomposition

Aggregation

Inheritence

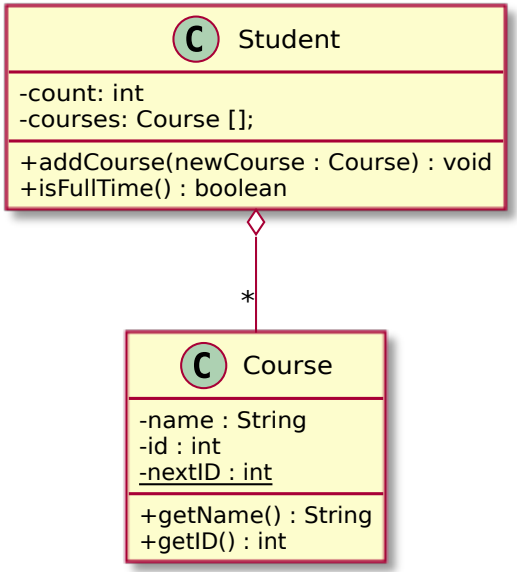
Inheritence Example

Refactoring our PBM reader

```
PBM() {
    Scanner sc =
        new Scanner(System.in);
    String dummy = sc.next();
    columns=sc.nextInt();
    rows=sc.nextInt();
    bits=new int[rows][columns];
    for (int i=0; i< rows; i++){
        String line=sc.next();
        for (int j=0; j<columns; j++)
            bits[i][j]=line.charAt(j)-'0';
    }
}
```

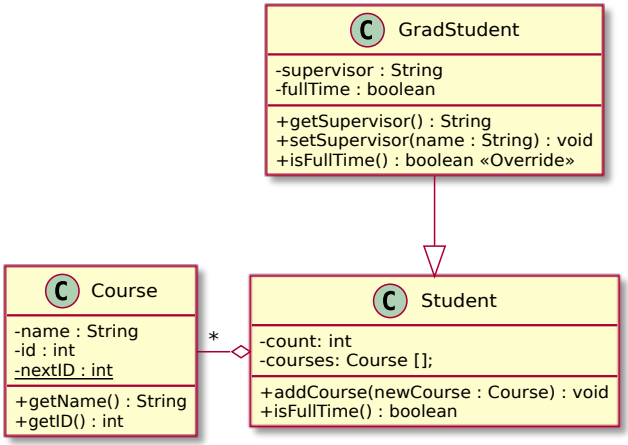
```
P1
3 3
111
101
111
```

Student aggregates Course




```
public class Student {
    ⋮
    Course [] courses;
    ⋮
}
```

GradStudent inherits from Student




```
public class GradStudent
    extends Student {
⋮
}
```

Overriding

```
public class Student {
    public boolean isFullTime(){
        return (count >= 3);
    }
    ⋮
}
```

Overriding

```
public class Student {
    public boolean isFullTime(){
        return (count >= 3);
    }
    ⋮
}

public class GradStudent
    extends Student {
    private boolean fullTime;
    public boolean isFullTime(){
        return fullTime;
    }
}
```

Method decomposition

Aggregation

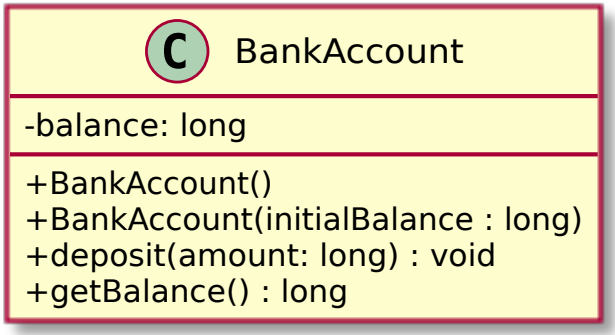
Inheritence

Inheritence Example

Bank Account Class

```
interest = (long)(account.getBalance()*INTEREST_RATE/100);
account.deposit(interest);
System.out.println("Balance after year 1 is ¢"
                   + account.getBalance());
```

BankAccount

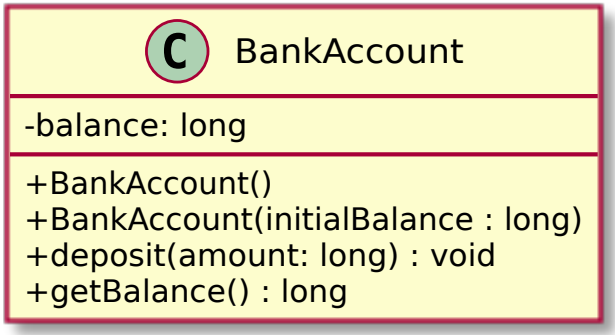


- Why `long`?
- How can we make this class more convenient to use?

Bank Account Class

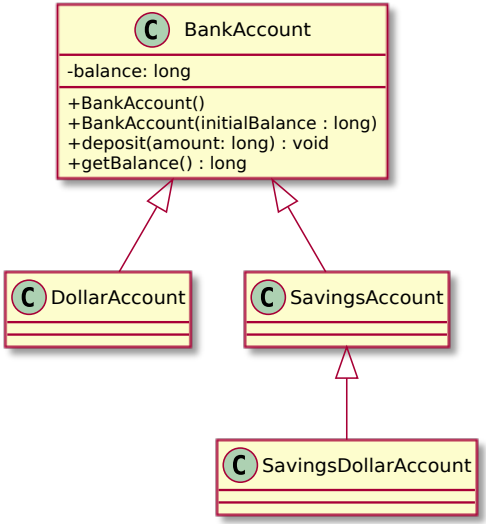
```
interest = (long)(account.getBalance()*INTEREST_RATE/100);
account.deposit(interest);
System.out.println("Balance after year 1 is ¢"
                   + account.getBalance());
```

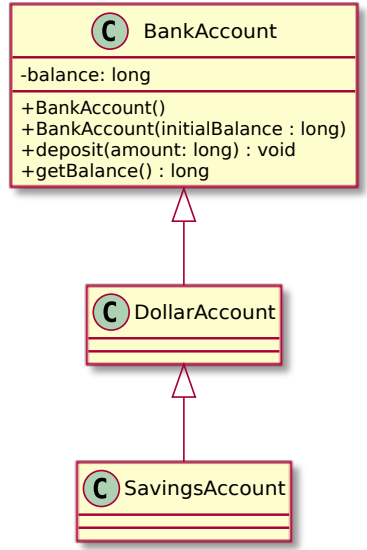
BankAccount



- Why `long`?
- How can we make this class more convenient to use?

Inheritence options





Limitations of overriding

DollarAccount

```
public class DollarAccount extends BankAccount {
    static private long toCents(long units, int cents) {
        return 100*units+cents;
    }
    public DollarAccount(long units, int cents) {
        super(toCents(units,cents));
    }
    public void deposit(long units, int cents) {
        super.deposit(toCents(units,cents));
    }
    // what happens if we try to override getBalance?
    public BigDecimal dollarBalance() {
        BigDecimal cents =
            new BigDecimal(super.getBalance());
        return cents.scaleByPowerOfTen(-2);
    }
```

A new beginning

```
public class DecimalAccount  {
    private BigDecimal balance;

    public DecimalAccount(long dollars, int cents)    {
        balance = new BigDecimal(dollars).
            add(new BigDecimal(cents).
                scaleByPowerOfTen(-2));
    }
    public void deposit(BigDecimal amount) {
        balance = balance.add(amount);
    }
    public BigDecimal getBalance() {
        return balance;
    }
    ⋮
```

DecimalAccount

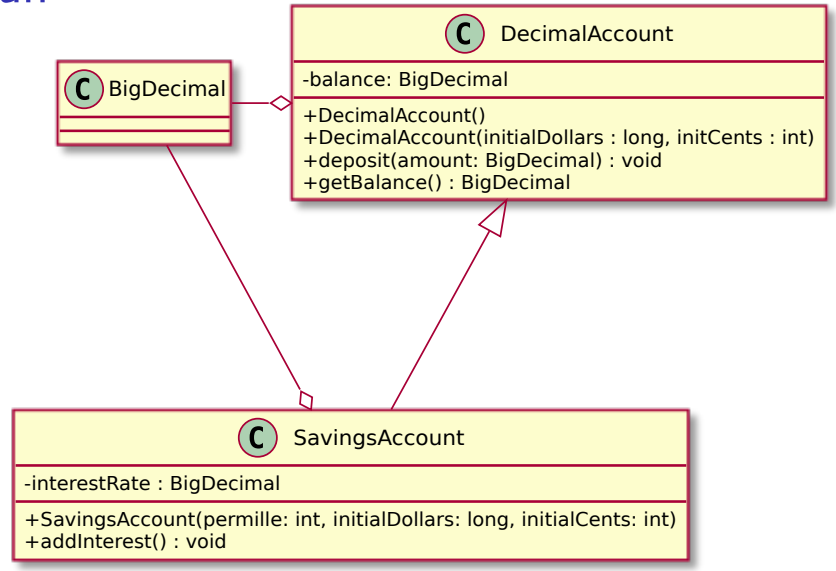
Using our new class

```
public static void main(String[] args)      {
    DecimalAccount account = new DecimalAccount(100,00);
    final BigDecimal INTEREST_RATE = new BigDecimal("0.05");
    BigDecimal interest;
    // compute and add interest for one period
    interest = account.getBalance().
        multiply(INTEREST_RATE);
    account.deposit(interest);
    System.out.println("Balance after year 1 is $"
                       + account.getBalance());
    ⋮
}
```

DecimalAccount

- how to improve toString()?

New Plan


DecimalAccount
-balance: BigDecimal
+DecimalAccount()
+DecimalAccount(initialDollars : long, initCents : int)
+deposit(amount: BigDecimal) : void
+getBalance() : BigDecimal
BigDecimal
SavingsAccount
-interestRate : BigDecimal
+SavingsAccount(permille: int, initialDollars: long, initialCents: int)
+addInterest() : void

SavingsAccount

```
public class SavingsAccount extends DecimalAccount {

    private bigDecimal interestRate;

    public SavingsAccount(int permille, long initialDollars
        super(initialDollars, initialCents);
        interestRate = new BigDecimal(permille).scaleByPowe
    }

    public void addInterest() {
        BigDecimal interest = getBalance().multiply(interes
        deposit(interest);
    }
}
```

SavingsAccount

Using SavingsAccount

SavingsAccount

```
public static void main(String[] args)
{
    SavingsAccount account = new SavingsAccount(50,
                                                100,00);

    // compute and add interest for one period
    account.addInterest();
    System.out.println("Balance after year 1 is $"
                       + account.getBalance());

    // add interest again
    account.addInterest();
    System.out.println("Balance after year 2 is $"
                       + account.getBalance());
}
```

Extra: Formatting BigDecimal

FormatTest

```
public static void main(String [] args) { //
    final BigDecimal INTEREST_RATE =
        new BigDecimal("0.05");
    DecimalAccount account = new DecimalAccount(1299,98);
    System.out.println(account.getBalance());
    BigDecimal interest =  account.getBalance().
        multiply(INTEREST_RATE);

    System.out.println(interest);
    System.out.println(interest.stripTrailingZeros());
    String s= interest.
        setScale(2,java.math.BigDecimal.ROUND_HALF_UP);
    System.out.println(s);
     System.out.println(NumberFormat.getCurrencyInstance().
                        format(interest));
}
```