CS133: Developing Programming Principles

Lecture 7

Encapsulation, Programmer Interface vs Implementation, Helper Methods, Overloading Methods

Encapsulation

- Separation of the **implementation** from the **programmer interface**.
- This is an extension of information hiding.

Programmer interface

- All the information needed for the programmer to use the class.
- I.e., method signatures, constants, pre and postconditions.
- Method bodies and other implementation details are **not** required!

`CashRegister` interface

```
// post: Initializes a register with
// balance of 0 and no sales.
public CashRegister()

// pre: initialBalance >= 0.00
// post: Initializes a register with
// initial balance initialBalance
public CashRegister(double initialBalance)

// post: returns number of sales completed
// so far
public int getNumSales()
```

`CashRegister` interface (2)

```
// post: returns number of items sold so far
public int getNumItemsSold()

// post: returns amount of money in this
// register's balance
public double getBalance()

// post: returns amount of revenue from all
//   completed sales
public double getRevenue()

// pre: amount >= 0.0
// post: adds amount to current balance
public void addToBalance(double amount)
```


`CashRegister` interface (3)

```
// pre: price >= 0.0, description != null
// post: adds the item of the given price to
// the current sale
public void scanItem(double unitCost,
     String description)

// pre: price >= 0.0, description != null,
//   quantity >= 0
// post: adds quantity items of the given
//   price to the current sale
public void scanItem(int quantity,
     double unitCost, String description)
```

`CashRegister` interface (4)

```
// post: returns amount owed in current sale
public double getSubTotal()

// pre: payment >= this.getSubTotal()
// post: pays for current sale from payment
//   and adds amount of current sale
//   to balance.
//   returns the amount of change from
//   the sale.
public double completeSale(double payment)
```


`CashRegister` interface (5)

```
// post: returns String representation of
// this cash register
public String toString()

// pre: other != null
// post: returns true iff this and other
// have the same number of sales, same
// balance, and same revenues.
public boolean equals(CashRegister other)
```

ATM class programmer interface

- Digits
- Deposit
- Withdrawal
- Confirm

Implementation

- The actual code of the methods that makes them work.
- **`private`** methods.
- Instance variables.

`CashRegister` variables

- **`private int numSales;`**
- **`private int numItemsSold;`**
- **`private double totalRevenue;`**
- **`private double balance;`**
- **`private double currRevenue;`**
- **`private int currNumItems;`**

Exercise: implement some methods

Helper methods

- Methods that perform subtasks of a bigger method.
- Should always be **`private`**.
- Make program shorter and easier to read and understand.
- Reuse code without copying andpasting.

Why encapsulate?

- Preserves the integrity of your code by blocking other programmers from modifying it to work in unplanned ways.
- Makes it easier for another programmer to use the object without worrying about the way it works.

Defining a well-encapsulated class

- Place an introductory comment before the beginning of the class that describes its functionality.
- All instance variables should be **`private`**.
- Provide accessor and mutator methods as appropriate.
- Make any helper methods **`private`**.
- Make sure pre and postconditions are clear and complete.

Overloading

- We have already seen an example of overloading in the form of + for **`String`** objects and numeric primitives.
- The same can be applied to methods.
- A class can have more than one method of the same name.

Overloading continued

- Each method **must** have either a different number or different kind of parameters.
- Examples:

```
public void scanItem(double unitCost,
      String description)
public void scanItem(int quantity,
      double unitCost, String description)

public String substring(int start)
public String substring(int start, int end)
```

Cannot overload return type

- The following is **not** allowed:

```
public int age(int y, int m, int d)
{ … }
public double age(int y, int m, int d)
{ … }
```

- If two methods have the same name, their parameter lists must be different.

Overloading: when?

- Two methods in a class should be overloaded only if:
  - They have the same basic task.
  - They need different information to accomplish that task.

Summary

- Encapsulation
- Programmer interface
- Implementation
- Overloading methods

ERROR: undefined
OFFENDING COMMAND:

STACK: