CSE 142 Summer 2001

Objects with Data

Introduction

- Review:
  - Classes and instances
  - methods, parameters, and arguments
- Today:
  - Objects that hold data – instance variables
  - Updating object data
  - Methods with a result value

Bank Accounts

- Suppose we wanted to create an object to represent a bank account (checking or savings)
  - What sort of messages (operations) would we want it to be able to do?
  - [new] What sort of information needs to be stored in the bank account object?

Bank Accounts

- Design your bank account here

BankAccount Notes

Using a Bank Account Object

- Draw the picture

```
BankAccount checking =
    new BankAccount( );
checking.deposit(1000.00);
checking.deposit(500.00);
double currentBalance =
    checking.getBalance( );
System.out.println(currentBalance);
```

- Picture

Instance Variables

- Fields like "balance" in a BankAccount object are called *instance variables*
  - Each new instance of the class (object) has its own set
- Pattern for a class declaration (enhanced)

```
class <class name> {
    <instance variable declarations>
    <method declarations>
}
```


Creating a BankAccount Class

- First attempt:

```
class BankAccount {
    double  balance = 0.0;          // account balance in dollars
    int     accountNumber = 0;      // account number
    String  accountName = ""        // account owner's name

    /** Deposit money in the account
     * @param amount amount of money to add to the account  */
    public void deposit(double amount) { ... }
    ...
}
```

- Something slightly bogus: the values given to accountNumber and accountName – we'll fix that later

Value-Returning Methods

- One of the methods in BankAccount is *getBalance*, which returns the current balance in the account

```
class BankAccount {
    double  balance = 0.0;          // account balance in dollars
    ...
    /** Return current account balance
     * @return current balance in dollars */
    public double getBalance( ) {
       return this.balance;
     }
    ...
}
```


Value-Returning Methods: Details

- The *type* of the value returned appears in the method declaration

```
public double getBalance( ) {
```

- A method that has a value must execute a *return* statement

```
return <expression> ;
```

This does two things

- Specifies the value that is returned (the <expression>)
- Terminates the method's execution and returns to the calling statement immediately

- A value-returning method can be used anywhere an expression of that type is expected

```
double currentBalance = checking.getBalance( );
```


this

- Recall: We select object fields (methods or instance variables) using a "."

```
<object name> . <field name>
```

- Inside a method, we can refer to fields of the current object with the keyword this

```
return this.balance;
```

- "this.balance" means the instance variable belonging to the current BankAccount

```
double checkingBalance = checking.getBalance( );
double savingsBalance = savings.getBalance( );
```


Mystery Explained

- When we were drawing trees and houses, we didn't use *this* explicitly, but we could have.

```
class Scene {
    public void drawScene( ) {
       this.drawHouse( );          // the drawHouse method associated with
                                   //   this particular Scene instance
       this.drawTree(180, 220, ...);  // etc.
       this.drawTree(250, 350, ...);
    }
    ...
}
```

Updating Instance Variables

- What about the account name and number? We'd like to be able to store a sensible value in these.

```
/** Store a new name in this BankAccount object
 * @param newAccountName the new name of this account
 */
public void setAccountName(String newAccountName) {
    this.accountName = newAccountName;
}
```

- New: *assignment statement.* Pattern

```
<existing name> = <expression with new value>      // bind name to new value
```

  - Notice the difference from a declaration that creates a new name.

```
<type> <new name> = <expression>;      // creates a new name & binds it
```


Deposit

- Update the account balance

```
/** Deposit money in this account
 * @param amount amount of money to deposit in dollars
 */
public void deposit(double amount) {
   this.balance = this.balance + amount;
}
```

- Execution of the assignment statement

(1) Evaluate the expression to the right of =
  ! All names in the expression must already have a value (why?)

(2) Bind the (existing) name on the left to the new value

- No problem if the name on the left also appears in the expression
- Question: What is the effect of

```
this.balance = this.balance + 1;
```


Summary

- A lot of big ideas here
  - Objects containing data – instance variables
  - Value-returning (non-void) methods
  - return statement
  - Assignment statement – updating instance variables
- Still to come
  - Initializing objects – constructors
  - Hiding object details – public vs private