CSE 142 Summer 2001

Constructors & Information Hiding

Introduction

- Review
  - Objects with data
  - Instance variables
  - Updating object data
  - Methods with result values
- Today
  - Constructors: initialization
  - public/private: information hiding

A BankAccount Class (Review)

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

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

- What is wrong with this picture?

Initial Values for Instance Variables (1)

- Wanted: some mechanism for giving instance variables sensible values in each a new object (instance)
- We can specify initial values when we declare (create) the instance variables

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

- Good idea? Why or why not?

Initial Values for Instance Variables (2)

- We could declare the instance variables without initial values, and provide methods to set the fields

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

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

- Good idea? Why or why not?

Initial Values for Instance Variables (3)

- We could supply a method the programmer could call to initialize all of the instance variables

```
class BankAccount {
    double  balance;                // account balance in dollars
    int     accountNumber;          // account number
    String  accountName;            // account owner's name
    /** Set the name, number, and balance for this account…. */
    public void initialize(String accountName int accountNumber,
                                        double balance) {
       this.accountName = accountName;
       this.accountNumber = accountNumber;
       this.balance = balance;
    }
    ...
}
```

Using method initialize

```
// create new checking account and initialize
BankAccount checking = new BankAccount( );
checking.initialize("Bill Gates", 1, 30000000000.17);
```

- Good idea? Why or why not?

Creating Objects with Initial Values

- We've already done this:

```
Rectangle rect = new Rect(100,50,250,30);
Rectangle anotherOne = new Rect(50,100,100,100,Color.blue,true);
Line segment = new Line(100,100, 200,100);
```

- How can we do this for our own objects (classes)?

Constructors

- A constructor is a special sort of method
  - Method name is the same as the class name
  - No result type (and no void)
  - Called automatically whenever an instance of the class is created

```
class BankAccount {
    …
    /** construct new BankAccount with given account number, name,
     * and initial balance. */
    public BankAccount(String accountName int accountNumber, double balance) {
       this.accountName = accountName;
       this.accountNumber = accountNumber;
       this.balance = balance;
    }
```


Multiple Constructors (1)

- A class can contain more than one constructor
  - Must be some difference in number or types of parameters
  - Constructor with matching parameter list called automatically when an instance of the class is created
  - Technical term: method (or constructor) *overloading*

```
class BankAccount {
    …
    /** construct new BankAccount with given account number, a balance
     * of 0, and a name that's a null string. */
    public BankAccount( ) {
       this.accountName = "";
       this.accountNumber = 0;
       this.balance = 0.0;
    }
```


Multiple Constructors (2)

```
class BankAccount {
    …
    /** construct new BankAccount with given account number, a balance
     * of 0, and a name that's a null string. */
    public BankAccount( ) { …}
    /** construct new BankAccount with given account number, name,
     * and initial balance. */
    public BankAccount(String accountName int accountNumber, double balance) { … }
}
```

- Which constructor is called?

```
BankAccount checking = new BankAccount("Bill Gates", 1, 30000000000.17);
BankAccount savings   = new BankAccount( );
```


Field Access

- Instance variables are also object fields
  - So we can do things like this

```
BankAccount leaky = new BankAccount("Life Savings", 1001, 12846.55);
leaky.deposit(100.0);
leaky.balance = leaky.balance - 100000.00;
leaky.accountNumber = -1;
```

- Good idea? Why or why not?

Information Hiding

- We would like to "protect" instance variables so they can only be changed, or even accessed by appropriate methods in the class
  - Reduce chances of corrupting instance data
  - Minimize scope of bugs (if an instance variable has a bogus value, the problem can be localized to methods in the class)
- Mechanism: declare instance variables with private access

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

    /** construct new BankAccount ... . */
    public BankAccount(String accountName int accountNumber, double balance) {
       this.accountName = accountName;         // ok
       ...
    }
}
```


Field Access Revisited

- If the BankAccount instance variables are private, which of these are legal? Why or why not?

```
BankAccount safe = new BankAccount("Life Savings", 1001, 12846.55);
safe.deposit(100.0);
safe.balance = safe .balance - 100000.00;
double amount = safe .withDraw(100000.00);
safe.accountNumber = -1;
```


public vs private

- Any member of a class (method or instance variable) can be specified public or private
  - public: member is accessible to any client code that can access the class or its instances
  - private: member is accessible only to methods and code inside the class
  - If neither, access is basically public
- Guidelines (good practices)
  - public: methods that are part of the class interface
  - private: all instance variables and any methods that are not part of the class interface

Accessor Functions

- If client code needs access to an instance variable, provide methods to allow this

```
class BankAccount {
    private double  balance;          // account owner's name
    ...
    /** return account balance... . */
    public String getAccountName( ) { return this.balance; }
    /** set account balance ... */
    public void setAccountName(double balance) {
       this.balance = balance;
    }
}
```

- Good idea? Why or Why Not?

Private Methods

- If a method is used in the class implementation, but is not part of the interface, it should be private (Why?)

```
class BankAccount {
    ...
    /** construct new BankAccount ... . */
    public BankAccount( ) { this.initialize("", 0, 0.0); }
     /** construct new BankAccount ... . */
    public BankAccount(String accountName int accountNumber, double balance) {
       this.initialize(accountName, accountNumber, balance); }

    // set this account's name, account number, and balance
    private initialize(String accountName, int accountNumber, double balance) {
       this.accountName = accountName;
       ....
    }
}
```


Summary

- Constructors
  - Guaranteed initialization when instances are created
  - Multiple definitions with different parameters lists possible
- Information hiding
  - public/private
  - Robustness, security