Implementing Classes: Black Boxes & Abstraction

Black Boxes

- A black box magically does its thing
- Hides its inner workings
- **Encapsulation:** the hiding of unimportant details
- What is the right *concept* for each particular black box?

Black Boxes

- Concepts are discovered through abstraction
- **Abstraction:** taking away inessential features, until only the essence of the concept remains
- In *object-oriented programming* the black boxes from which a program is manufactured are called objects

Levels of Abstraction: A Real-Life Example

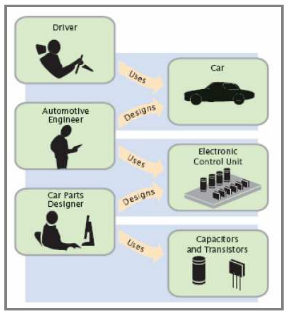

Figure 1: Levels of Abstraction in Automobile Design

Levels of Abstraction: A Real- Life Example

- Users of a car do not need to understand how black boxes work
- Interaction of a black box with outside world is **well-defined**
  - Drivers interact with car using pedals, buttons, etc.
  - Mechanic can test that engine control module sends the right firing signals to the spark plugs
  - For engine control module manufacturers, transistors and capacitors are black boxes magically produced by an electronics component manufacturer

Levels of Abstraction: A Real- Life Example

- Encapsulation leads to efficiency:
  - Mechanic deals only with car components (e.g. electronic control module), not with sensors and transistors
  - Driver worries only about interaction with car (e.g. putting gas in the tank), not about motor or electronic control module

Levels of Abstraction: Software Design

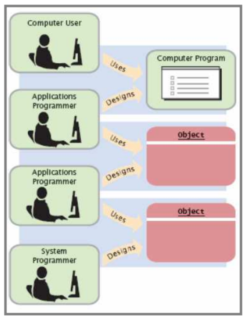

Figure 2: Levels of Abstraction in Software Design

Levels of Abstraction: Software Design

- Old times: computer programs manipulated primitive types such as numbers and characters
- Manipulating too many of these primitive quantities is too much for programmers and leads to errors
- *Solution: Encapsulate routine computations to software black boxes*

Levels of Abstraction: Software Design

- Abstraction used to invent higher-level data types
- In object-oriented programming, objects are black boxes
- **Encapsulation:** Programmer using an object knows about its behavior, but not about its internal structure

Levels of Abstraction: Software Design

- In software design, you can design good and bad abstractions with equal facility; understanding what makes good design is an important part of the education of a software engineer
- First, define behavior of a class; then, implement it

Question?

- Suppose you are working in a company that produces personal finance software. You are asked to design and implement a class for representing bank accounts. Who will be the users of your class?
- **Answer:** Other programmers who work on the personal finance application

Designing the Public Interface of a Class

Designing the Public Interface of a Class

- Behavior of bank account (abstraction):
  - deposit money
  - withdraw money
  - get balance

Designing the Public Interface of a Class: Methods

- Methods of `BankAccount` class:

```
deposit
withdraw
getBalance
```

- We want to support method calls such as the following:

```
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
System.out.println(harrysChecking.getBalance());
```

Designing the Public Interface of a Class: Method Definition

- access specifier (such as `public`)
- return type (such as `String` or `void`)
- method name (such as `deposit`)
- list of parameters (`double amount` for `deposit`)
- method body in `{ }`

Designing the Public Interface of a Class: Method Definition

Examples

```
public void deposit(double amount) { . . . }
public void withdraw(double amount) { . . . }
public double getBalance() { . . . }
```

Syntax 3.1: Method Definition

```
accessSpecifier returnType methodName(parameterType
 parameterName, . . .)
{
   method body
}
```

Example:

```
   public void deposit(double amount)
   {
      . . .
   }
```

Purpose:
To define the behavior of a method

Designing the Public Interface of a Class: Constructor Definition

- A constructor initializes the instance variables
- Constructor name = class name

```
public BankAccount()
{
   // body--filled in later
}
```

Designing the Public Interface of a Class: Constructor Definition

- Constructor body is executed when new object is created
- Statements in constructor body will set the internal data of the object that is being constructed
- All constructors of a class have the same name
- Compiler can tell constructors apart because they take different parameters

Syntax 3.2: Constructor Definition

```
accessSpecifier ClassName(parameterType parameterName, . . .)
{
    constructor body
}
```

Example:

```
public BankAccount(double initialBalance)
{
   . . .
}
```

Purpose:
To define the behavior of a constructor

`BankAccount` Public Interface

- The public constructors and methods of a class form the *public interface* of the class.

```
public class BankAccount
{
   // Constructors
   public BankAccount()
   {
      // body--filled in later
   }
   public BankAccount(double initialBalance)
   {
      // body--filled in later
   }
   // Methods
   public void deposit(double amount)
```

`BankAccount` Public Interface

```
   {
      // body--filled in later
   }
   public void withdraw(double amount)
   {
      // body--filled in later
   }
   public double getBalance()
   {
      // body--filled in later
   }
   // private fields--filled in later
}
```

Syntax 3.3: Class Definition

```
accessSpecifier class ClassName
{
   constructors
   methods
   fields
}
```

Example:

```
public class BankAccount
{
   public BankAccount(double initialBalance) { . . . }
   public void deposit(double amount) { . . . }
  . . .
}
```

Purpose:
To define a class, its public interface, and its implementation details