University of British Columbia
CPSC 111, Intro to Computation
Jan-Apr 2006

Tamara Munzner

Class Design

Lecture 6, Tue Jan 24 2006

based on slides by Paul Carter

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

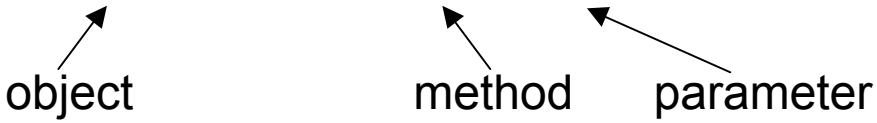
Reading This Week

- Chap 3

Recap: Methods and Parameters

- Methods are how objects are manipulated
  - pass information to methods with parameters
    - inputs to method call
    - tell `charAt` method which character in the String object we're interested in
  - methods can have multiple parameters
    - API specifies how many, and what type
  - two types of parameters
    - explicit parameters given between parens
    - implicit parameter is object itself

```
String firstname = "Alphonse";
char thirdchar = firstname.charAt(2);
```



Recap: Return Values

- Methods can have return values
- Example: **`charAt`** method result
  - return value, the character 'n', is stored in **`thirdchar`**

```
String firstname = "kangaroo";
char thirdchar = firstname.charAt(2);
```

return value object method parameter

- Not all methods have return values
  - No return value indicated as `void`

Recap: Constructors and Parameters

- Many classes have more than one constructor, taking different parameters
  - use API docs to pick which one to use based on what initial data you have

Constructor Summary

`String()`
Initializes a newly created `String` object so that it represents an empty character sequence.

`String(String original)`
Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

```
animal = new String();
animal = new String("kangaroo");
```

Recap: Keyboard Input

- Want to type on keyboard and have Java program read in what we type
  - store it in variable to use later

- Scanner class does the trick
  - **`java.util.Scanner`**
  - nicer than **`System.in`**, the analog of **`System.out`**

Recap: Importing Packages

- Collections of related classes grouped into packages
  - tell Java which packages to keep track of with import statement
  - again, check API to find which package contains desired class
- No need to import String, System.out because core **`java.lang`** packages automatically imported

Recap: Scanner Class Example

```
import java.util.Scanner;

public class Echo
{
    public static void main (String[] args)
    {
        String message;
        Scanner scan = new Scanner (System.in);
        System.out.println ("Enter a line of text: ");
        message = scan.nextLine();
        System.out.println ("You entered: \""
                            + message + "\"");
    }
}
```

- Print out the message on the display

Escape Characters

- How can you make a String that has quotes?
  - `String foo = “oh so cool”;`
  - `String bar = “oh so \”cool\”, more so”;`
- Escape character: backslash
  - general principle

Objectives

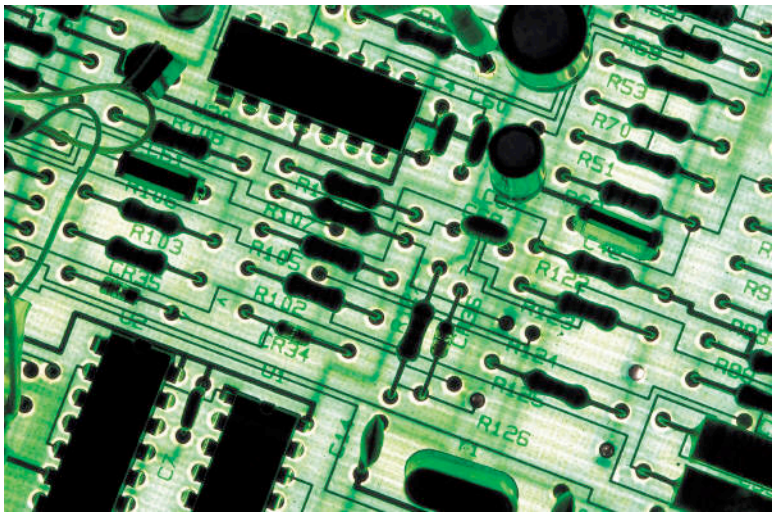
- understand principles of abstraction and encapsulation
- understand how to design new classes using these principles
- understand how to implement new classes in Java

Creating Classes and Objects

- So far you've seen how to use classes created by others
- Now let's think about how to create our own
- Example: rolling dice
  - doesn't exist already in Java API
  - we need to design
  - we need to implement
- Start with two design principles

Abstraction

- Abstraction: process whereby we
  - hide non-essential details
  - provide a view that is relevant
- Often want different layers of abstraction depending on what is relevant



Encapsulation

- Encapsulation: process whereby
  - inner workings made inaccessible to protect them and maintain their integrity
  - operations can be performed by user only through well-defined interface.
  - aka information hiding
- Cell phone example
  - inner workings encapsulated in hand set
    - cell phone users can't get at them
  - intuitive interface makes using them easy
    - without understanding how they actually work

Approach

- Apply principles of abstraction and encapsulation to classes we design and implement
  - same idea as examples from daily life
  - only in software

Designing `Die` Class

- Blueprint for constructing objects of type **`Die`**
- Think of manufacturing airplanes
  - build one blueprint
  - manufacture many instances from it
- Consider two viewpoints
  - client programmer: want to use **`Die`** object in a program
  - designer: creator of **`Die`** class

Client Programmer

- What operations does client programmer need?
  - what methods should we create for `Die`?

Designer

- Decide on inner workings
  - implementation of class
- Objects need state
  - attributes that distinguish one instance from another
  - many names for these
    - state variables
    - fields
    - attributes
    - data members
  - what fields should we create for `Die`?

Information Hiding

- Hide fields from client programmer
  - maintain their integrity
  - allow us flexibility to change them without affecting code written by client programmer
    - Parnas' Law:
      - "Only what is hidden can by changed without risk."

Public vs Private

- **`public`** keyword indicates that something can be referenced from outside object
  - can be seen/used by client programmer
- **`private`** keyword indicates that something cannot be referenced from outside object
  - cannot be seen/used by client programmer
- Let's fill in public/private for **`Die`** class

Public vs. Private Example

```
Die myDie = new Die();

myDie.                          //not allowed!
```

Unified Modeling Language

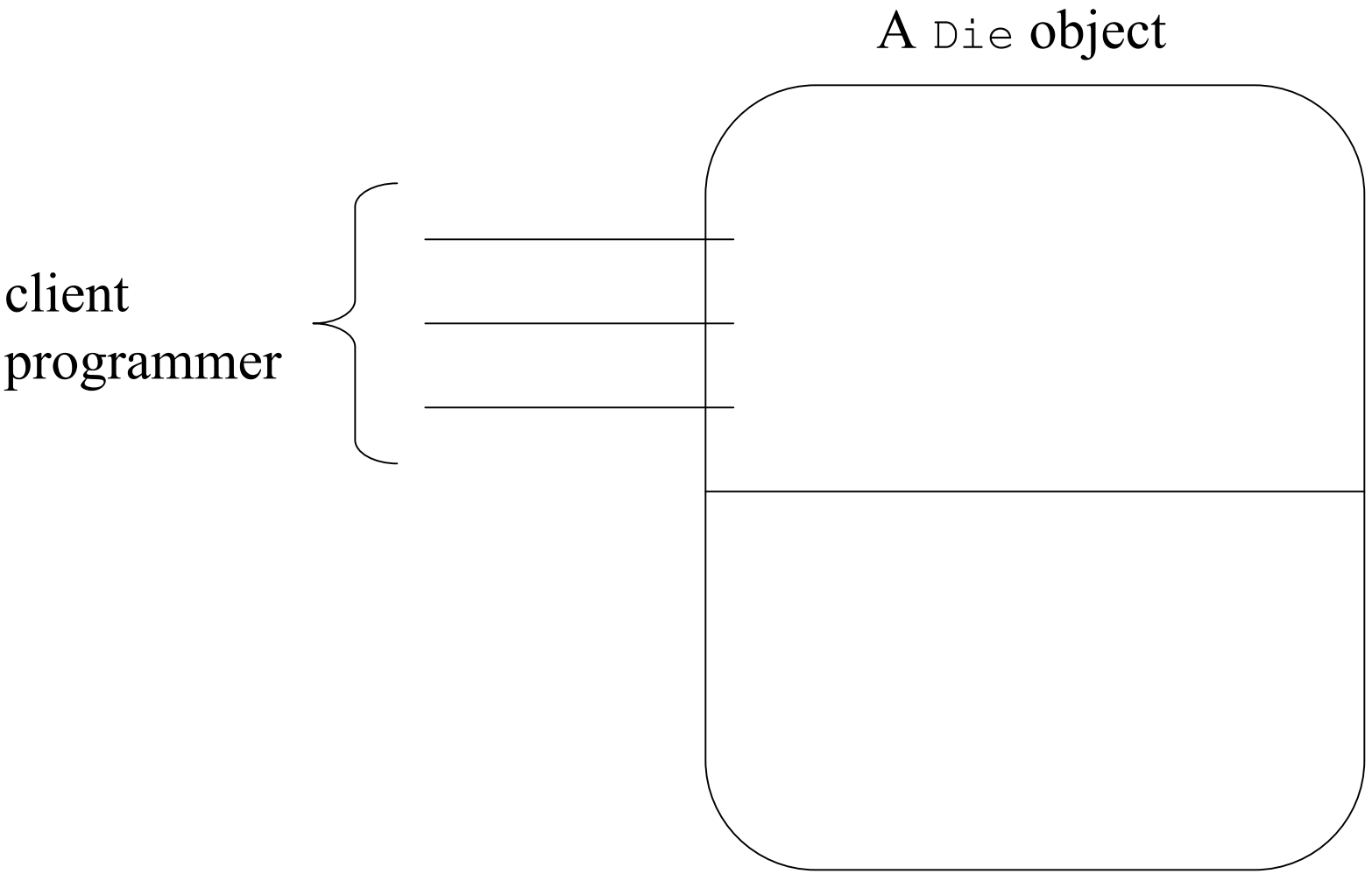
- Unified Modeling Language (UML) provides us with mechanism for modeling design of software
  - critical to separate design from implementation (code)
  - benefits of good software design
    - easy to understand, easy to maintain, easy to implement
- What if skip design phase and start implementing (coding)?
  - code difficult to understand, thus difficult to debug
- We'll use UML class diagrams represent design of our classes
- Once the design is completed, could be implemented in many different programming languages
  - Java, C++, Python,...

UML for `Die`

- UML diagram representing **`Die`** class design

Encapsulation Diagram

- Illustrate principle of encapsulation for **`Die`**



Implementing `Die`

```
public class Die
{




}
```

Implementing `RollDice`

```
public class RollDice
{
   public static void main ( String [] args)
   {









}
```