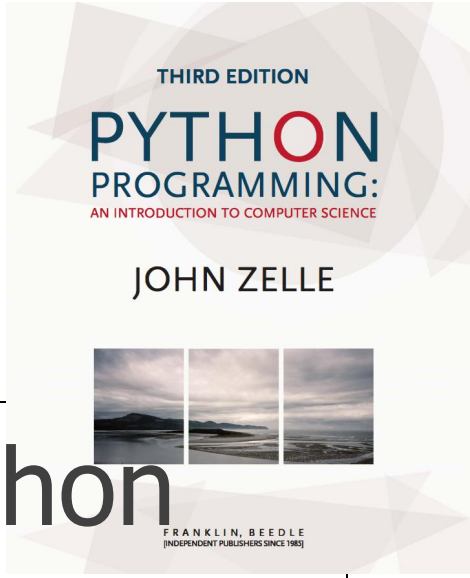


An Introduction to Python Programming

Chapter 12: Object-Oriented Design

Objectives

- To understand the **process** of **object-oriented design**.
- To understand the concepts of **encapsulation**, **polymorphism** and **inheritance** as they pertain to object-oriented design and programming.
- To be able to design moderately complex software using object-oriented design.

The Process of OOD

- The **essence of OOD** is describing a system in terms of **magical black boxes** and **their interfaces**.
  - Interfaces：methods
  - User/Client
  - Black boxes：Objects

The Process of OOD

- Here are some **guidelines** for OOD:
  - **Look for object candidates** to define a set of objects that will be helpful in solving the problem.

×

√

  - **Identify instance variables.** Some object attributes will have primitive values; others might themselves be complex types that suggest other useful objects/classes.

The Process of OOD

```
def __init__(self):
    self.age = 10
```

```
class People:
  def __init__(self,age=1):
      self.age = age
```

```
class Dog:
    kind = 'canine'         # class variable shared by all instances
    def __init__(self, name):
        self.name = name    # instance variable unique to each instance
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                  # shared by all dogs
'canine'
>>> e.kind                  # shared by all dogs
'canine'
>>> d.name                  # unique to d
'Fido'
>>> e.name                  # unique to e
'Buddy'

'''
```

- **Think about interfaces.** Services are provided via interfaces.Think about what operations would be required for objects of that class.**All of the object's data** should be manipulated through your methods.

The Process of OOD

- **Refine the nontrivial methods.** Accomplished with a couple of lines of code? Require to develop an algorithm? Some new interactions with other classes are needed?

- **Design iteratively.**

- **Try out alternatives.**You won't really know how a system should be built until you've already built it the wrong way.

- **Keep it simple.**

Case Study: Racquetball Simulation

- Inputs:
  - Probability for player A
  - Probability for player B
  - n
- Output:
  - Results

- **Consider shutouts:** When one player gets to 7 points the other player has 0, the game ends.

Case Study: Racquetball Simulation

- First，find a set of **objects** to represent a single game of racquetball.
  - Track information about players ------ ***skill levels***
  - It needs to play it ----- ***play method***

```
theGame = RBallGame(probA, probB)
theGame.play()
```

Case Study: Racquetball Simulation

- Then turn our attention to collecting statistics.
  - wins for A, wins for B, shutouts for A, and shutouts for B.
  - Group them into a single object,referring to a class ***SimStats.***
    - ***initial***
    - ***update***

Case Study: Racquetball Simulation

- Our **main function**.

```
def main():
    printIntro()
    probA, probB, n = getInputs()
    # Play the games
    stats = SimStats()
    for i in range(n):
        theGame = RBallGame(probA, probB) # create a new game
        theGame.play()                    # play it
        stats.update(theGame)             # get info about completed game
    # Print the results
    stats.printReport()
```

Case Study: Racquetball Simulation

- Implementing **SimStats**

```
class SimStats:
    def __init__(self):
        self.winsA = 0
        self.winsB = 0
        self.shutsA = 0
        self.shutsB = 0
```

- About **update method :**

The final score of the game is ?

Directly access the instance variables of **aGame is not allowed**.

⟶ **getScores**

Case Study: Racquetball Simulation

```
def update(self, aGame):
    a, b = aGame.getScores()
    if a > b:                                   # A won the game
        self.winsA = self.winsA + 1
        if b == 0:
            self.shutsA = self.shutsA + 1
    else:                                       # B won the game

        self.winsB = self.winsB + 1
        if a == 0:
            self.shutsB = self.shutsB + 1
```

Case Study: Racquetball Simulation

- We can complete the SimStats class by writing ***printReport*** method

```
def printReport(self):
    # Print a nicely formatted report
    n = self.winsA + self.winsB
    print("Summary of", n , "games:\n")
    print("          wins (% total)   shutouts (% wins)  ")
    print("-----------------------------------------")
    self.printLine("A", self.winsA, self.shutsA, n)
    self.printLine("B", self.winsB, self.shutsB, n)
```

Case Study: Racquetball Simulation

- ***printLine*** ：A good start is to define a template for the information that will appear in each line:

```
def printLine(self, label, wins, shuts, n):
    template = "Player {0}:{1:5}  ({2:5.1%}) {3:11}   ({4})"
    if wins == 0:          # Avoid division by zero!
        shutStr = "-----"
    else:
        shutStr = "{0:4.1%}".format(float(shuts)/wins)
    print(template.format(label, wins, float(wins)/n, shuts, shutStr))
```

Case Study: Racquetball Simulation

- Turn our attention to ***RballGame***
  - What to do ?
    - ***constructor***
    - ***play***
    - ***getScores***
  - What to know ?
    - ***the probability for each player***
    - ***the score for each player***
    - ***which player is serving***

particular player's properties

a property of the game

New Classes ← ***New Objects***

Case Study: Racquetball Simulation

- The **constructor** for RBallGame:

```
class RBallGame:
    def __init__(self, probA, probB):
        self.playerA = Player(probA)
        self.playerB = Player(probB)
        self.server = self.playerA  # Player A always serves first
```

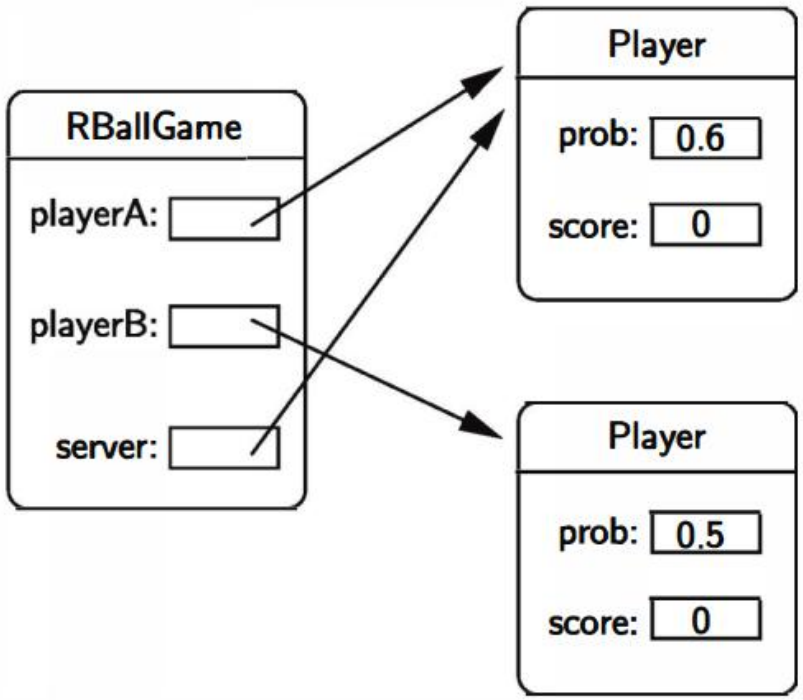
Case Study: Racquetball Simulation

- The **constructor** for RBallGame:

```
class RBallGame:
    def __init__(self, probA, probB):
        self.playerA = Player(probA)
        self.playerB = Player(probB)
        self.server = self.playerA  # Player A always serves first


theGame = RBallGame(.6,.5)
```

Case Study:
Racquetball Simulation



Case Study: Racquetball Simulation

- **How to play it?**

serve rallies,award points,until the game is over

changeServer method	***incScore*** method	one loop; ***isOver*** method
to the game	***to the server***	***to the game***

Case Study:
Racquetball Simulation

```
def play(self):
    while not self.isOver():
        if self.server.winsServe():
            self.server.incScore()
        else:
            self.changeServer()
```

- About **getScores**:
  Actually,only the player objects know the scores

```
def getScores(self):
    return self.playerA.getScore(), self.playerB.getScore()
```

add to the player

Case Study: Racquetball Simulation

- **Implementing Player**

a constructor , winsServe, incScore, getScore

```
def __init__(self, prob):
    # Create a player with this probability
    self.prob = prob
    self.score = 0

def winsServe(self):
    return random() < self.prob
```

Case Study: Racquetball Simulation

```
def incScore(self):
    self.score = self.score + 1

def getScore(self):
    return self.score
```

Those pieces are so simple that their implementations are obvious!

*Let's go to **P430** to see out complete program!*

Case Study:
Dice Poker

- **Program Specification**

Write a game program that allows a user to play video poker using dice.

Case Study:
Dice Poker

- The basic set of **rules**

① The player starts with $100.
② Each round costs $10,subtracted from the player's money at the start.
③ All five dice are rolled randomly .
④ The player gets two chances to enhance the hand by rerolling some or all of the dice.
⑤ At the end of the hand, the player's money is updated.

Case Study:
Dice Poker

The interface's characteristics:

① The current score is displayed.
② Automatically terminates when the player goes broke.
③ Choose to quit at appropriate points.
④ Present visual cues:what is going on?What the valid user responses are?

Case Study:
Dice Poker

The payout schedule

hand	pay
Two Pairs	$ 5
Three of a Kind	$ 8
Full House (A Pair and a Three of a Kind)	$ 12
Four of a Kind	$ 15
Straight (1–5 or 2–6)	$ 20
Five of a Kind	$ 30




Case Study:
Dice Poker

- Identifying **Candidate Objects.**

- the ***Dice*** class
  - constuctor
  - rollAll 、 roll
  - values
  - score

- A **PokerApp** object
  - the current amount of money
  - the dice
  - the number of rolls, etc.
  - run,etc.

Case Study:
Dice Poker

- A **PokerInterface** object
  - get information from the user
  - display information about the game

- **Implementing the Model**

  - Implementing the lower-level ***Dice*** class

    changing numbers , roll selected dice(all) , return values(payment)

    → **use a list of five ints**

Case Study:
Dice Poker

```
class Dice:
    def __init__(self):
        self.dice = [0]*5
        self.rollAll() # The code is set to some random values.
    def roll(self, which): # for example roll ([0 ,3 ,4] )
        for pos in which:
            self.dice[pos] = randrange(1,7)
    def rollAll(self):
        self.roll(range(5))
    def values(self):
        return self.dice[:]
  # This will not affect the original copy stored in the Dice object.
```

Case Study: Dice Poker

- About the ***score*** method : a multi-way decision

```
def score(self):
    # Create the counts list
    counts = [0] * 7
    for value in self.dice:
        counts[value] = counts[value] + 1

    # score the hand
    if 5 in counts:
        return "Five of a Kind", 30
    elif 4 in counts:
        return "Four of a Kind", 15
    elif (3 in counts) and (2 in counts):
        return "Full House", 12
```

Case Study:
Dice Poker

```
elif 3 in counts:
    return "Three of a Kind", 8
elif not (2 in counts) and (counts[1]==0 or counts[6] == 0):
    return "Straight", 20
elif counts.count(2) == 2:
    return "Two Pairs", 5
else:
    return "Garbage", 0
```

counts

0	1	2	3	4	5	6

NOTICE: If the dice are: [3 ,2 ,5 ,2 ,3] then the count list would be [0 ,0 ,2 ,2 ,0 , 1 ,0] .

Case Study:

Dice Poker

- Implementing **PokerApp:**
  keep **track** of the **dice**, the amount of **money** and some **user interface**.

run the program， play another hand

run method　　user interface***;playRound*** method

(handles the scoring aspect)

Case Study:
Dice Poker

```
def run(self):
    while self.money >= 10 and self.interface.wantToPlay():
        self.playRound()
    self.interface.close()
def playRound(self):
    self.money = self.money - 10
    self.interface.setMoney(self.money)
    self.doRolls()
    result, score = self.dice.score()
    self.interface.showResult(result, score)
    self.money = self.money + score
    self.interface.setMoney(self.money)
```

Case Study:
Dice Poker

#We need a loop that continues rolling user-selected dice until either the user chooses to quit rolling or the limit of three rolls is reached.

```
def doRolls(self):
    self.dice.rollAll()
    roll = 1
    self.interface.setDice(self.dice.values())
    toRoll = self.interface.chooseDice()
    while roll < 3 and toRoll != []:
        self.dice.roll(toRoll)
        roll = roll + 1
        self.interface.setDice(self.dice.values())
        if roll < 3:
            toRoll = self.interface.chooseDice()
```

Case Study: Dice Poker

- **A Text-Based UI**
  - ***PokerInterface*** class.
    setMoney, setDice, showResult , wantToPlay , chooseDice.
  - **Graphical interfaces** are usually more complicated than a **text-based interface**.
  - We can **tweak** the **`PokerApp`** class so that the user interface is supplied as a **parameter** to the constructor.

```
class PokerApp:
    def __init__(self, interface):
        self.dice = Dice()
        self.money = 100
        self.interface = interface
```

Case Study: Dice Poker

- The ***TextInterface*** class is shown on P441

- Here is a **complete program**

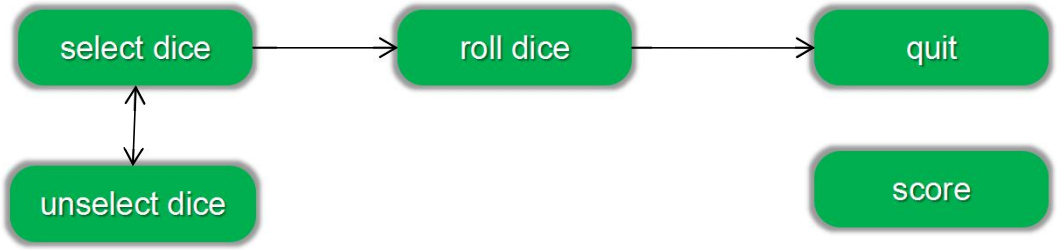
```
# textpoker.py -- video dice poker using a text-based

from pokerapp import PokerApp
from textpoker import TextInterface

inter = TextInterface()
app = PokerApp(inter)
app.run()
```

Case Study: Dice Poker

- **Developing a GUI**
  - Designing the **Interaction**
    - setDice
    - setMoney
    - showResult
    - handle the status bar



Case Study:
Dice Poker

- Requirements
  - ① Showing the user **which dice** are **currently selected**
  - ② Need a good way for the user to indicate that they wish to **stop rolling**.

"Roll Dice" button with no dice selected

A "Score" button

Case Study: Dice Poker

- How it looks?

  The exact layout of the widgets:

a list of Buttons, DieView classes

Case Study:
Dice Poker

- **Choose** method of the **Pokerinterface class**:
  When inputing ,only the valid buttons will be set to **active** .

```
def choose(self, choices):
    buttons = self.buttons

    # activate choice buttons, deactivate others
    for b in buttons:
        if b.getLabel() in choices:
            b.activate()
        else:
            b.deactivate()

    # get mouse clicks until an active button is clicked
    while True:
        p = self.win.getMouse()
        for b in buttons:
            if b.clicked(p):
                return b.getLabel()  # function exit here.
```

Case Study:
Dice Poker

- **DieView** class can be found on P383
  - For example,the ***setValue method*** :

```
def setValue(self, value):
    # Turn all the pips off
    for pip in self.pips:
        pip.setFill(self.background)

    # Turn the appropriate pips back on
    for i in self.onTable[value]:
        self.pips[i].setFill(self.foreground)
```

Case Study:
Dice Poker

- We need to add a **setColor method.**

```
change foreground to the new color
redraw the current value of the die
```

```
self.value = value
```

is added to setValue method

```
def setColor(self, color):
    self.foreground = color
    self.setValue(self.value) #redraw the die
```

Case Study: Dice Poker

- Creating the **Interface**

```
class GraphicsInterface:
    def __init__(self):
        self.win = GraphWin("Dice Poker", 600, 400)
        self.win.setBackground("green3")
        banner = Text(Point(300,30), "Python  Poker  Parlor")
        banner.setSize(24)
        banner.setFill("yellow2")
```

Case Study:
Dice Poker

```
banner.setStyle("bold")
banner.draw(self.win)
self.msg = Text(Point(300,380), "Welcome to the Dice Table")
self.msg.setSize(18)
self.msg.draw(self.win)
self.createDice(Point(300,100), 75)
self.buttons = []
self.addDiceButtons(Point(300,170), 75, 30)
b = Button(self.win, Point(300, 230), 400, 40, "Roll Dice")
self.buttons.append(b)
b = Button(self.win, Point(300, 280), 150, 40, "Score")
self.buttons.append(b)
b = Button(self.win, Point(570,375), 40, 30, "Quit")
self.buttons.append(b)
self.money = Text(Point(300,325), "$100")
self.money.setSize(18)
self.money.draw(self.win)
```

Case Study:
Dice Poker

```
def createDice(self, center, size):
    center.move(-3*size,0)
    self.dice = []
    for i in range(5):
        view = DieView(self.win, center, size)
        self.dice.append(view)
        center.move(1.5*size,0)

def addDiceButtons(self, center, width, height):
    center.move(-3*width, 0)
    for i in range(1,6):
        label = "Die {0}".format(i)
        b = Button(self.win, center, width, height, label)
        self.buttons.append(b)
        center.move(1.5*width, 0)
```

Case Study:
Dice Poker

- **Implementing the Interaction**

① Output methods:**setMoney** , **showResult , setDice**

```
def setMoney(self, amt):
    self.money.setText("${0}".format(amt))

def showResult(self, msg, score):
    if score > 0:
        text = "{0}! You win ${1}".format(msg, score)
    else:
        text = "You rolled {0}".format(msg)
    self.msg.setText(text)
```

Case Study:
Dice Poker

```
def setDice(self, values):
    for i in range(5):
        self.dice[i].setValue(values[i])
```

#It sets the ith die to show the ith value.

② Input methods: **wantToPlay** , **chooseDice** , **close**

```
def wantToPlay(self):
    ans = self.choose(["Roll Dice", "Quit"])
    self.msg.setText("")
    return ans == "Roll Dice"
```

Case Study:
Dice Poker

```
def chooseDice(self):
    # choices is a list of the indexes of the selected dice
    choices = []                    # No dice chosen yet
    while True:
        # wait for user to click a valid button
        b = self.choose(["Die 1", "Die 2", "Die 3", "Die 4", "Die 5",
                         "Roll Dice", "Score"])

        if b[0] == "D":             # User clicked a die button
            i = int(b[4]) - 1       # Translate label to die index
            if i in choices:        # Currently selected, unselect it
                choices.remove(i)
                self.dice[i].setColor("black")
            else:                   # Currently deselected, select it
                choices.append(i)
```

Case Study:
Dice Poker

```
        self.dice[i].setColor("gray")
else:                           # User clicked Roll or Score
    for d in self.dice:         # Revert appearance of all dice
        d.setColor("black")
    if b == "Score":            # Score clicked, ignore choices
        return []
    elif choices != []:         # Don't accept Roll unless some
        return choices          #   dice are actually selected
```

Programming Excercise

- Can you combine the classes、methods、functions in the Dice Poker game ？Debug it and modify to add some functions such as printing a nice introduction, providing help with the rules, and keeping track of high scores.