How to THINK like a Programmer

Problem Solving for the Bewildered

paul vickers

chapter 7

object orientation: taking a different view

www.cengage.co.uk/vickers

Purpose

- Looking at another way to think about structuring problems and solutions
- Seeing how we can think in terms of objects as the basis for algorithms
  - Recognize the difference between procedural and object-oriented problem solving
  - Analyze real-world problems to identify the object classes, properties, and methods needed to solve a problem in an object-oriented manner

Procedural programming

- So far our problem solving has been based in the **procedural paradigm**
  - Consider what actions are necessary
  - Find the data needed to support those actions
  - Supported by languages such as C, PHP, Perl, Pascal, Ada, COBOL, Fortran (pre 2003) etc
- Programming languages like Smalltalk, Java, C++, C#, Python (to name a few) support the **object oriented paradigm**

Thinking about data

- In procedural programming we think about the data belonging to the problem then design algorithms to process (manipulate) that data to achieve the desired outcome
- For making coffee we identified the following data items
  - The number of coffees to be made
  - The milk preference of a drinker
  - The sugar requirements of a drinker
  - The number of cups poured so far
  - The number of sugars added to a cup so far
- Then designed an algorithm to make and pour the required number of cups of coffee

Thinking about data

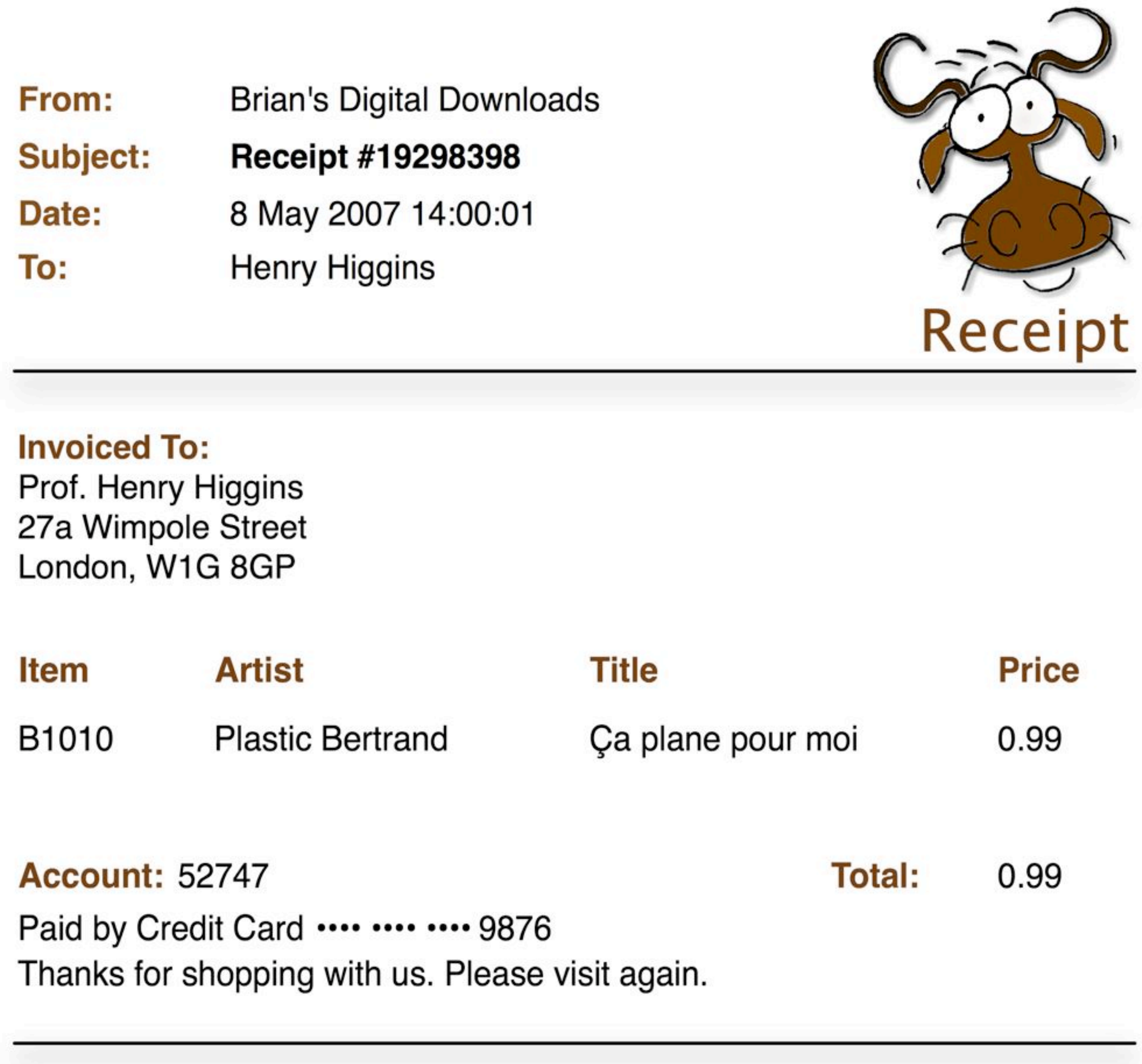
- Consider the problem of opening and operating an account with an online digital music download service.
- Identify key operations:
  - open account, credit account, spend money, download track, query the balance, query download history, close account, etc
- Identify key data:
  - account number, amount deposited, amount spent, current balance, customer name, customer address, country in which the account is held, date account was opened, date account was closed, etc
- Notice some data associated with account, some with customer, and so on

Thinking about objects

- In object oriented programming (OOP) we start by identifying the **objects** at work in our problem scenario
- We notice some of the data values are associated directly with the music store account, while others are associated with the customer who holds the account
  - The customer (name, address) pays money into the account
  - The account receives money from the customer
  - The account has a balance which the customer may request
  - The customer may view a history of all recent downloads
  - Receipts for track downloads are e-mailed to the customer
- We can think in terms of **objects**: ‘my iTunes account’, ‘Customer no. 34578’ etc

ACTIVITY

Consider the BriTunes statement below and identify the information that pertains to Professor Higgins' account



From:	Brian's Digital Downloads
Subject:	**Receipt #19298398**
Date:	8 May 2007 14:00:01
To:	Henry Higgins

Invoiced To:
Prof. Henry Higgins
27a Wimpole Street
London, W1G 8GP

Item	Artist	Title	Price
B1010	Plastic Bertrand	Ça plane pour moi	0.99

Account: 52747 **Total:** 0.99

Paid by Credit Card •••• •••• •••• 9876

Thanks for shopping with us. Please visit again.

BriTunes objects

- We see the BriTunes music store might have entities/objects called:
  - Account #52747
  - Customer Professor Henry Higgins
  - Item #B1010
  - etc
- There will be >1 account, so we say Account #52747 is an **instance** of a more general BriTunes Account **class**
- Higgins is an **instance** of the Customer **class**
- etc

Classes

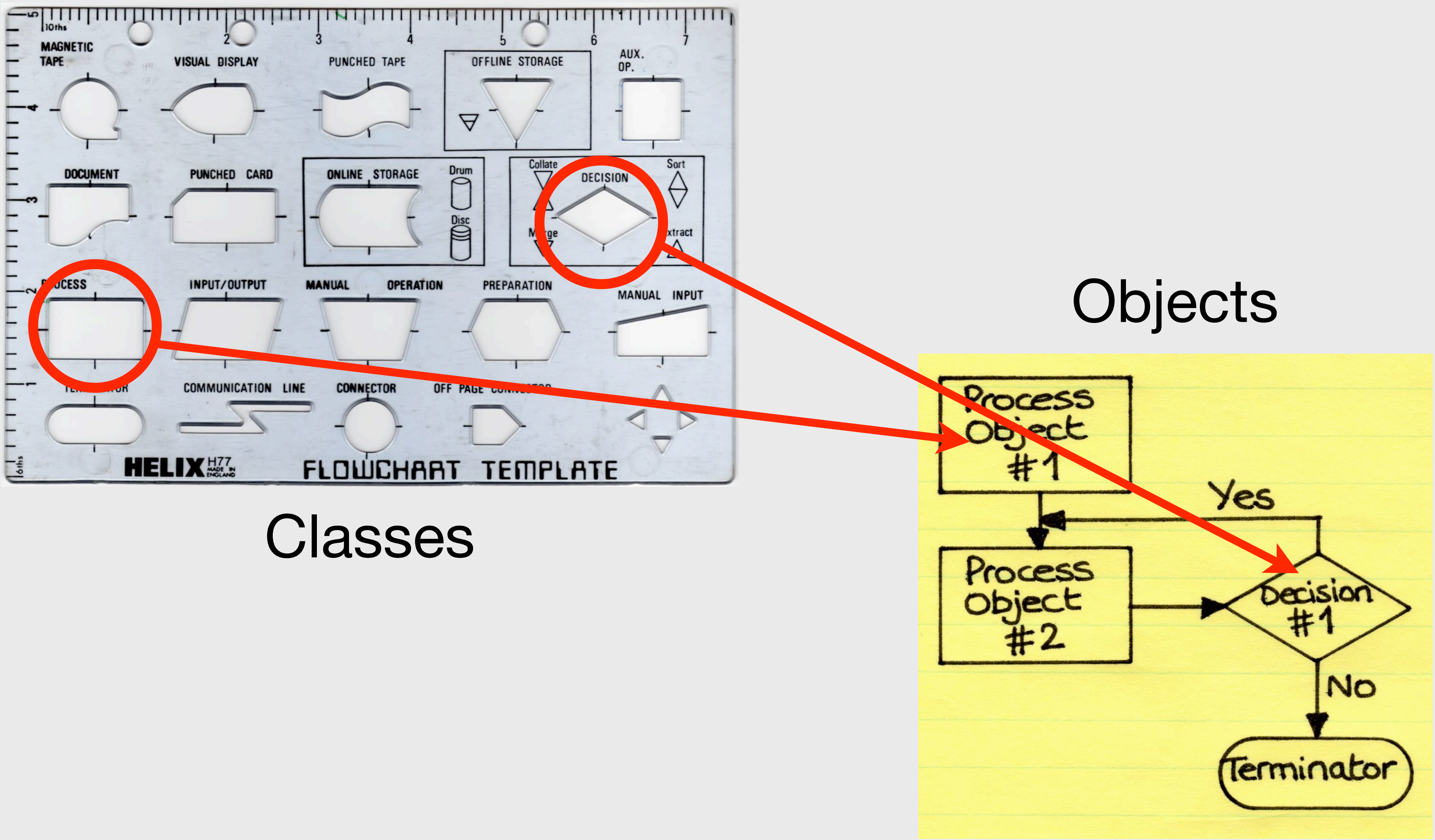
- In OOP a fundamental concept is the **class.**
  - a name given to a kind of **object**
  - defines the range of **properties** and behviours associated with objects belonging to that class
- An `Account` object will have
  - **properties**: balance, open date, no. downloads, customer number, etc
  - **behaviours**: open, receive funds, close, withdraw funds, etc

Everyday classes

- We use classes every day
- The number 7 belongs to the class of numbers known as natural numbers
  - **Properties**: natural numbers are in the range 1, 2, ..., ∞ and possess no fractional parts (they are whole numbers)
  - **Behaviours**: arithmetic can be performed upon natural numbers, so +, –, ×, ÷, etc
- In OOP, then, we could define a **class** called `NaturalNumbers`
  - Objects belonging to this class could only be assigned values between 1 and ∞, and could only have the defined arithmetic operations performed upon their values
- This reminds us of ADTs. Not the same, but similar

Classes: object template

- Think of a class as a template for stamping out individual objects



Classes

Objects

Classes & objects

- **Objects** are individual **instances** of a **class**
- The **class** defines:
  - the **properties** or **members**: i.e. the data items belonging to objects of that class
  - the **methods**: the algorithms that manipulate the properties
- The data (properties) belonging to an object may **only** be changed/accessed by methods belonging to that object
  - i.e. one object may not directly change the data values belonging to another object -- a class, therefore, must provide methods for updating data values

Getting up procedurally

- We can specify the procedural solution to the problem of getting up in the morning:

```
1.  Switch off alarm ;
2.  Get out of bed ;
3.  Wash/shower face, brush teeth, etc. ;
4.  Get dressed ;
```

- How to approach this in an OOP manner?
- What are the classes involved
  - the behaviours (methods)
  - the data (properties)

Getting up OOP style

class Person	
Properties	awake: yes, no ; inBed: yes, no ; needsShower: yes, no ; isDressed: yes, no ;
Methods	WakeUp ; GoToSleep ; GetUp ; GoToBed ; GetWashed ; GetDressed ; GetUndressed ;

ACTIVITY

In the light of what was said above about an object's operations being used to view and change an object's properties, why might it not be appropriate to have a `SwitchOffAlarm` operation in the `Person` class?

ACTIVITY

Following the way we defined a `Person` class above to guide you try writing out a class definition for an `Alarm` object. Think about the essential properties we need to capture for this problem and then consider what methods will be needed to change those properties.

Alarm class

class Alarm	
Properties	ringing: yes, no ; time: 00:00:00 to 23:59:00 ; alarmTime: 00:00:00 to 23:59:00 ; alarmIsSet: on, off ;
Methods	SetTime: hh:mm:ss ; GetTime ; SetAlarmTime: hh:mm ; GetAlarmTime ; SetAlarm ; UnsetAlarm ; StartRinging ; SwitchOff ; (i.e. stop ringing)

Controlling it all

- How to get `Person` and `Alarm` objects to do anything?
- Need a **controller** algorithm to orchestrate it all
- First we create a `Person` object called `Brian`

```
brian ← new Person ;
```

- Creating an instance of a class is called **instantiation**
  - like using a template to stamp out a new shape, cookie, etc
- The object `brian` now has data items `awake`, `inBed`, `needsShower`, `isDressed` and all the methods that also belong to all `Person` objects
- How to switch off the alarm?

The alarm

- Create an instance of the `Alarm` class

```
briansAlarm ← new Alarm;
```

- and tell it to switch off

```
tell briansAlarm ← SwitchOff ;
```

- Put it all together and we get

```
brian ← new Person;
briansAlarm ← new Alarm ;
tell brian WakeUp ;
tell briansAlarm SwitchOff ;
tell brian GetUp ;
tell brian GetWashed ;
tell brian GetDressed ;
```

ACTIVITY

Write the statements to set the time on Brian’s alarm clock

Setting the alarm

- We could set the alarm like this:

```
1.  brian ← new Person;
2.  briansAlarm ← new Alarm ;
3.  tell briansAlarm SetTime: currentTime ;
4.  tell briansAlarm SetAlarmTime: '07:00:00' ;
5.  tell brian WakeUp ;
6.  tell briansAlarm SwitchOff ;
7.  tell brian GetUp ;
8.  tell brian GetWashed ;
9.  tell brian GetDressed ;
```

- But when should action 6 be triggered? We need some sort of wait loop to wait until the alarm rings

Sleeping

- We could show the time during which Brian sleeps like this

```
1.  brian ← new Person;
2.  briansAlarm ← new Alarm ;
3.  tell briansAlarm SetTime: currentTime ;
4.  tell briansAlarm SetAlarmTime: '07:00:00' ;
5.  wait until briansAlarm ringing property = 'Yes' ;
6.  tell brian WakeUp ;
7.  tell briansAlarm SwitchOff ;
8.  tell brian GetUp ;
9.  tell brian GetWashed ;
10. tell brian GetDressed
```

- but statement #5 appears to be directly accessing a property of `briansAlarm`: we said before this is not allowed

Getting values out

- Lets introduce a new method to the Alarm class called RingingStatus which is used to tell the outside world whether the alarm is ringing or not
- Here is its algorithm

```
RingingStatus:
← ringing ;
```

- It has a single statement which sends out the value of the ringing property
- We can use this in the controller algorithm thus

```
tell briansAlarm RingingStatus: answer ;
```

- where answer is a variable belonging to the controller algorithm

Finished controller

- Here is the completed controller with a proper wait loop

```
1.  brian ← new Person;
2.  briansAlarm ← new Alarm ;
3.  tell briansAlarm SetTime: currentTime ;
4.  tell briansAlarm SetAlarmTime: '07:00:00' ;
5.  DO
      5.1.  tell briansAlarm RingingStatus: answer ;
    WHILE (answer ≠ 'Yes') ;
6.  tell brian WakeUp ;
7.  tell briansAlarm SwitchOff ;
8.  tell brian GetUp ;
9.  tell brian GetWashed ;
10. tell brian GetDressed ;
```

end of chapter 7