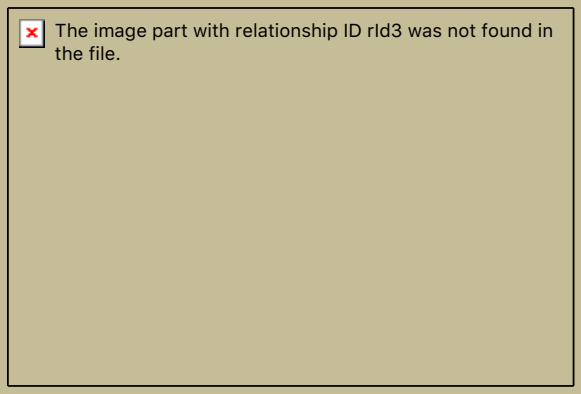
COMPUTER SCIENCE 1: STARTING COMPUTING CSCI 1300



Ioana Fleming / Vipra Gupta
Spring 2018
Lecture 22

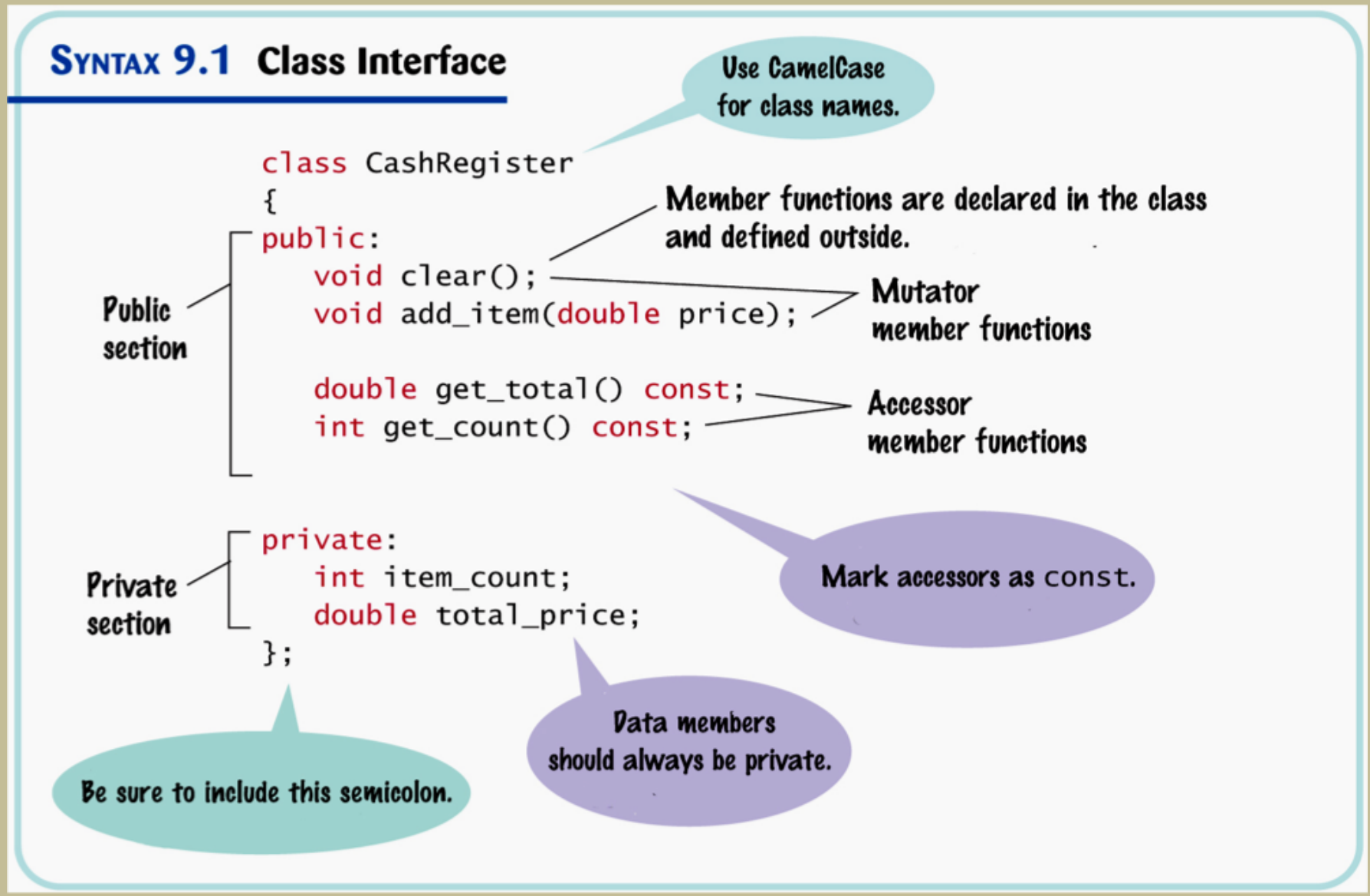
Announcements

- Rec 9 due on 3/17
- Hmwk 7 (Project 2) due on 3/25
- **Practicum 2: March 14th**, 2018
  - Loops: while, for
  - Strings
  - Arrays
  - File I/O
  - *sign up on Moodle to take the exam at 6pm*
- Practicum Review – Monday 3/12, 4 pm

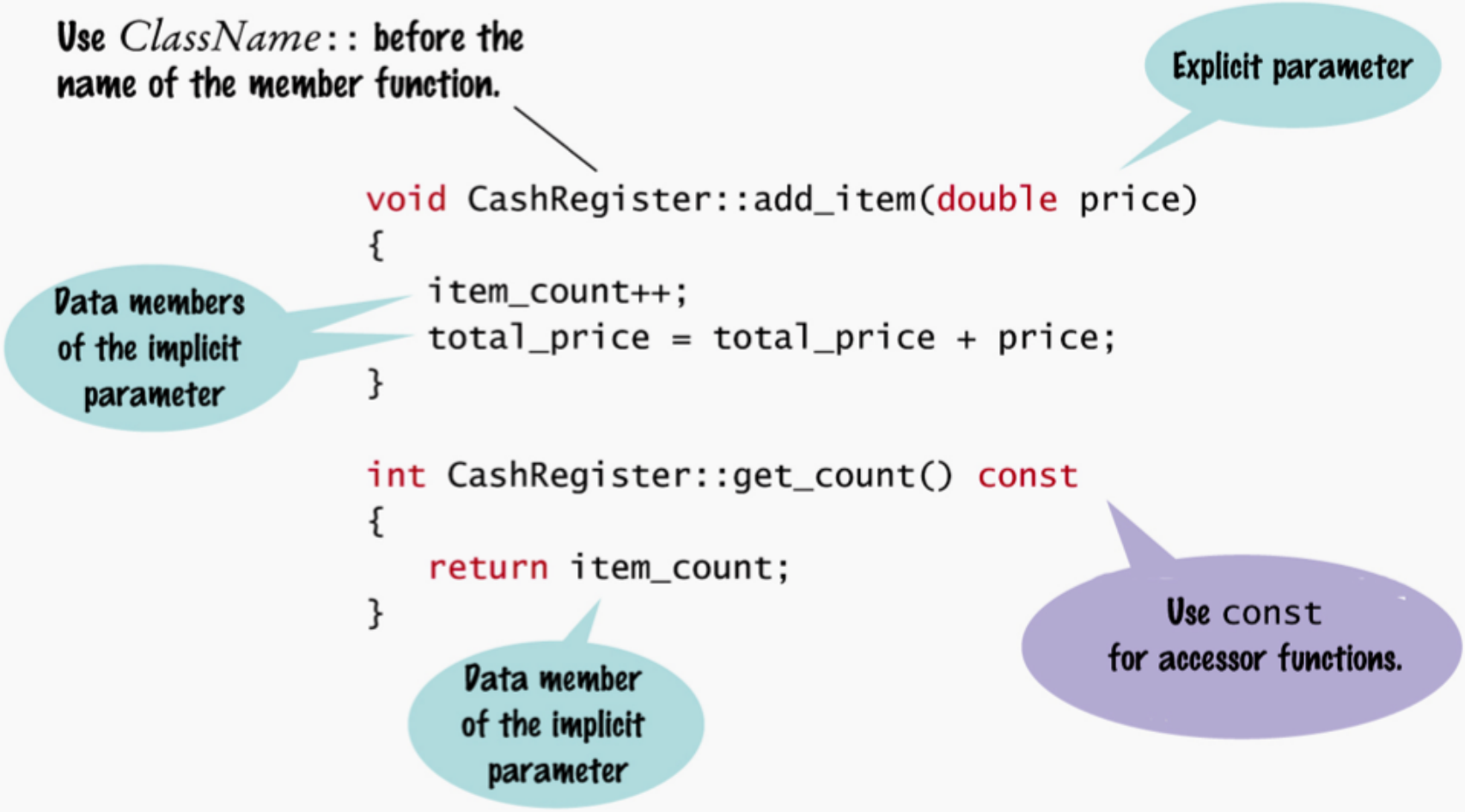
Agenda

- Today:
  - Object Oriented Programming
  - Classes and objects

Class Definition Syntax



Syntax 9.2 Member Function Definition



OOP – how/where do I write the class

- Option 1: everything in one file *main_program.cpp*
  - class interface, then member functions definitions, then main()
- Option 2: one class file + one driver file
  - MyClass.h contains class interface, then member functions definitions (**this one for Moodle questions)**
  - driver file *main_program.cpp* contains main()
    - in driver file: `#include "MyClass.h"`
- Option 3: two class files + one driver file
  - MyClass.h contains the class interface
  - MyClass.cpp contains member functions definitions
    - `#include "MyClass.h"`
  - driver file *main_program.cpp*
    - `#include "MyClass.h"`

Constructors

```
House house1;
House house2;
House house3;
...
```

A friendly construction worker
reading a class definition

Constructors

A *constructor* is a member function that *initializes* the data members of an object.

The constructor is automatically called whenever an object is created.

```
CashRegister register1;
```

(You don't see it but it's there.)

Constructors

By supplying a constructor,

you can ensure that all data members are properly set *before* any member functions act on an object.

(Ah, consistency …)

Constructors

By supplying a constructor,

you can ensure that all data members are properly set before any member functions act on an object.

What would be the value of a data member that was not (no way!) properly set?

GARBAGE

Constructors

"Garbage" is a ***technical*** computer science term.

It means…

…well…

"garbage."

Constructors

To understand the importance of constructors, consider the following statements:

```
CashRegister register1;
register1.add_item(1.95);
int count = get_count(); // May not be 1
```

Notice that the programmer forgot to call **`clear`** before adding items.

(Smells like “garbage” to me! – that previous value of item_count is “garbage”)

Constructors

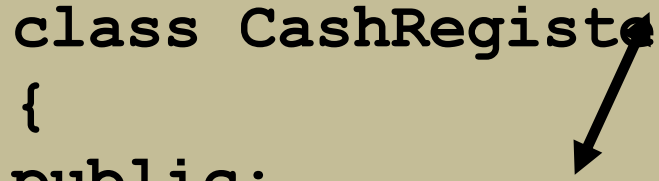
Constructors are written to guarantee that an object is always fully and correctly initialized *when it is defined*.

You declare constructors in the class definition:

```
class CashRegister
{
public:
   CashRegister(); // A constructor
...
};
```

Constructors

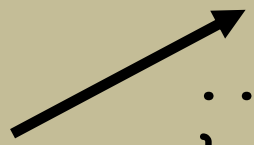
The name of a constructor is identical to the name of its class:



```
class CashRegister
{
public:
   CashRegister(); // A constructor
...
};
```

Constructors

There must be ***no*** *return type*, not even **`void`**.



```
class CashRegister
{
public:
   CashRegister(); // A constructor
...
};
```

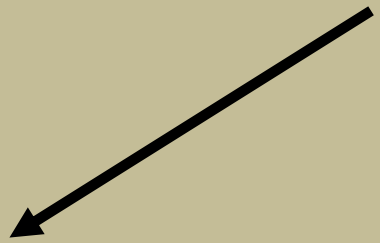
Constructors

And, of course, you must define the constructor.

```
CashRegister::CashRegister()
{
   item_count = 0;
   total_price = 0;
}
```

Constructors

To connect the definition with the class, you must use the same `::` notation

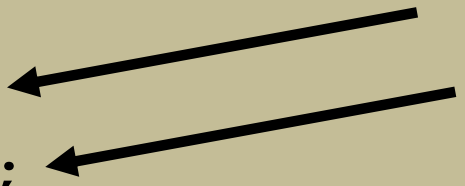


```
CashRegister::CashRegister()
{
   item_count = 0;
   total_price = 0;
}
```

Constructors

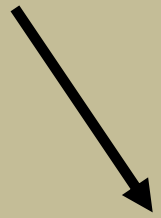
You should choose initial values
for the data members so the object is correct.

```
CashRegister::CashRegister()
{
   item_count = 0;
   total_price = 0;
}
```



Constructors

And still no return type.



```
CashRegister::CashRegister()
{
   item_count = 0;
   total_price = 0;
}
```

Constructors

A constructor with no parameters is called a *default constructor*.

```
CashRegister::CashRegister()
{
   item_count = 0;
   total_price = 0;
}
```

Constructors

Default constructors are called when you define an object and do not specify any parameters for the construction.

```
CashRegister register1;
```

Notice that you do NOT use an empty set of parentheses.

Constructors

`register1.item_count` and **`register1.total_price`** are set to zero as they should be.

```
CashRegister register1;
```

New Example

```
class BankAccount
{
public:
   BankAccount(); // A constructor
...


private:
   double balance;
...
};
```

Constructors

Constructors can have parameters,
and constructors can be *overloaded*:

```
class BankAccount
{
public:
   // Sets balance to 0
   BankAccount();
   // Sets balance to initial_balance
   BankAccount(double initial_balance);
   // Member functions omitted
private:
   double balance;
};
```

Constructors

When you construct an object, the compiler chooses the constructor that matches the parameters that you supply:

```
BankAccount joes_account;
   // Uses default constructor
BankAccount lisas_account(499.95);
   // Uses BankAccount(double) constructor
```

Constructors

It is good design to *think* about what
values you should put in
numeric and pointer data members (arrays).

They will be garbage if you don't set them in the constructor.

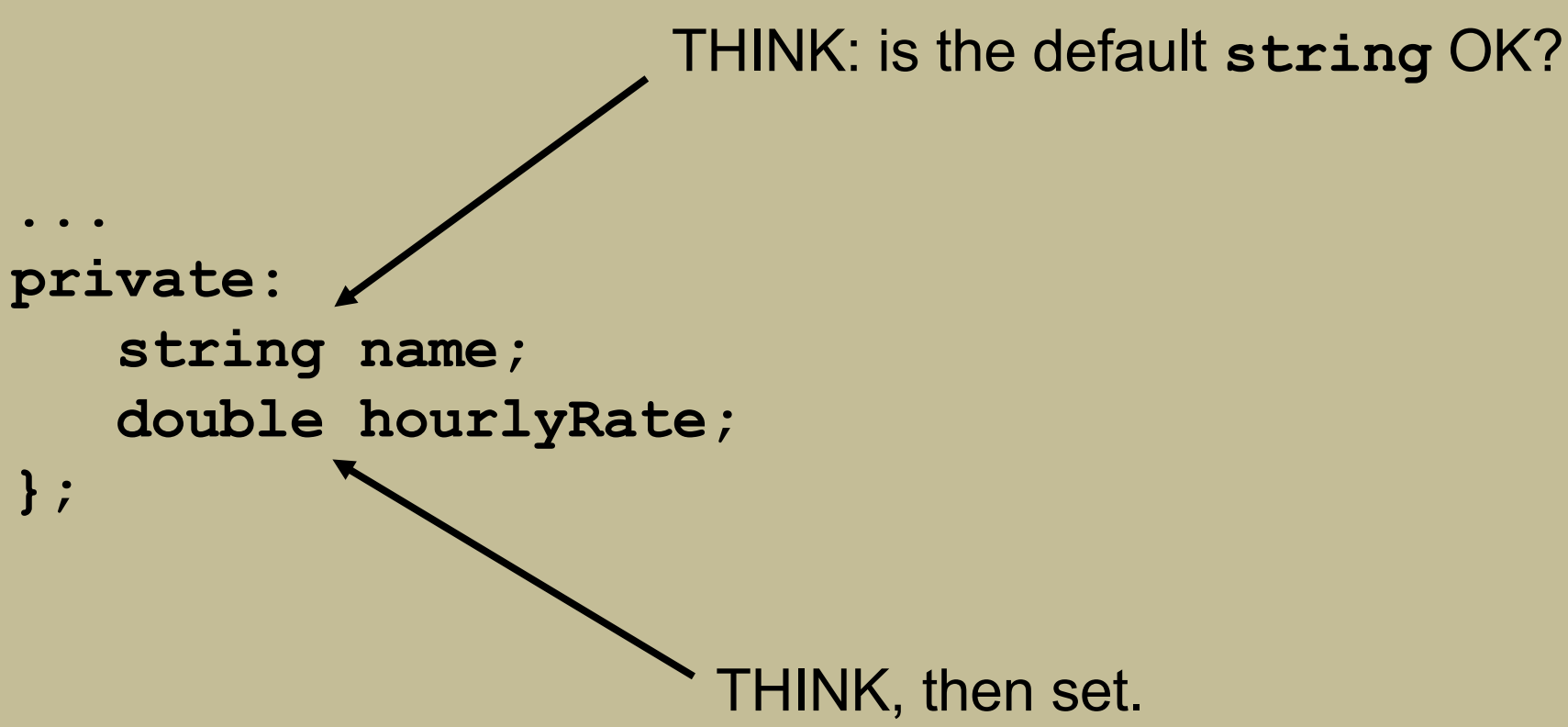
Is that OK?

Constructors

Data members of classes that have constructors will not be garbage.

For example, the **`string`** class has a default constructor that sets **`string`**s to the *empty string* ("").

Constructors



Common Error: Trying to Use the Constructor to Reset

You cannot use a constructor to “reset” a variable.

It seems like a good idea but you can’t:

```
CashRegister register1;
...
register1.CashRegister(); // Error
```

Constructors – The System Default Constructor

If you write no constructors at all,
the compiler automatically generates
a system default constructor
that initializes all data members of
class type with their default constructors

(which is just garbage for numeric and array data members).

Tracing Objects

Recall how you hand traced code
to help you understand functions.

Adapting tracing for objects
will help you understand objects.

Grab some index cards
(blank ones).

Tracing Objects

You know that the **`public:`** section is for others.
That's where you'll write methods for their use.

```
class CashRegister
{
public:
   void clear();
   void add_item(double price);
   double get_total() const;
   int get_count() const;
private:
   int item_count;
   double total_price;
};

...

CashRegister reg1;
```

That will be the front of the card.

Tracing Objects

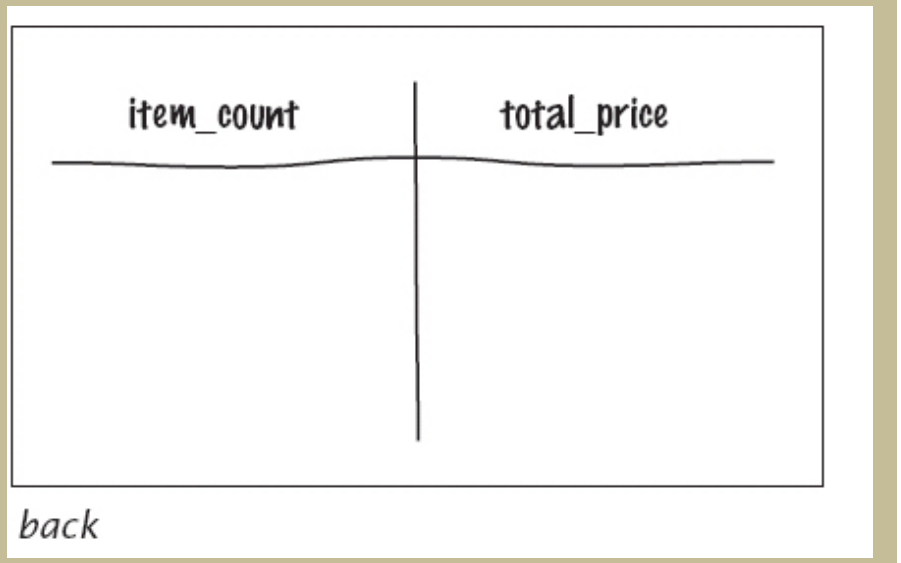
You know that the **`private:`** section is for your data – they are not allowed to mess with it except through the public methods you provide.

```
class CashRegister
{
public:
   void clear();
   void add_item(double price);
   double get_total() const;
   int get_count() const;
private:
   int item_count;
   double total_price;
};

...

CashRegister reg1;
```

That will be the back of the card.



Tracing Objects

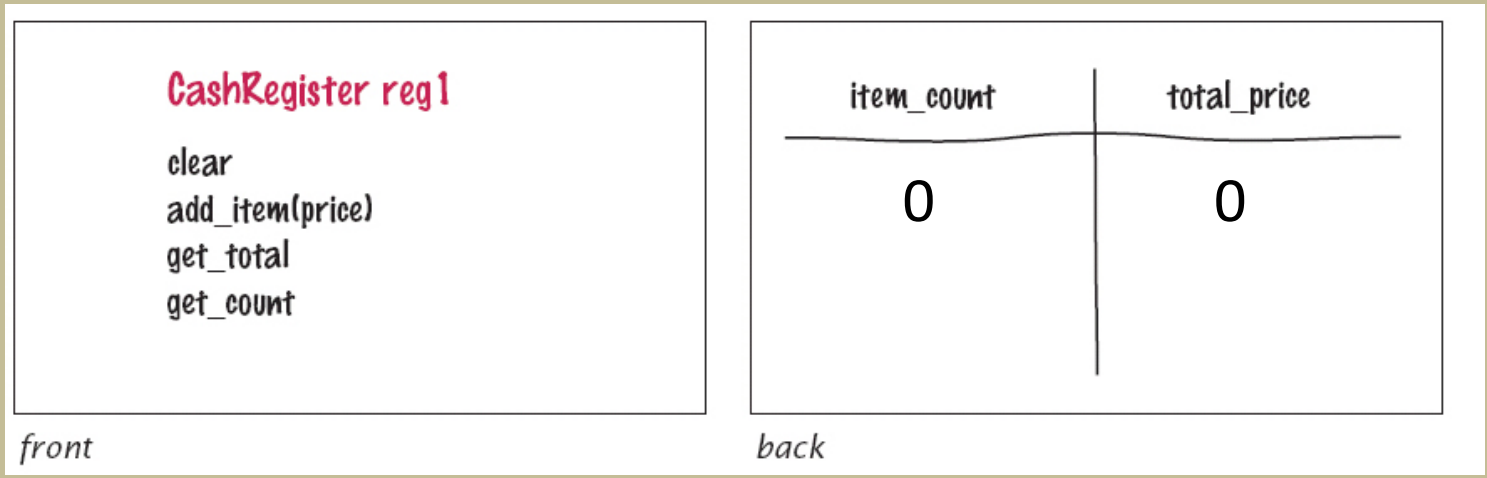
You'll need a card for every variable.

You might want to make several now.

Tracing Objects

```
CashRegister reg1;
```

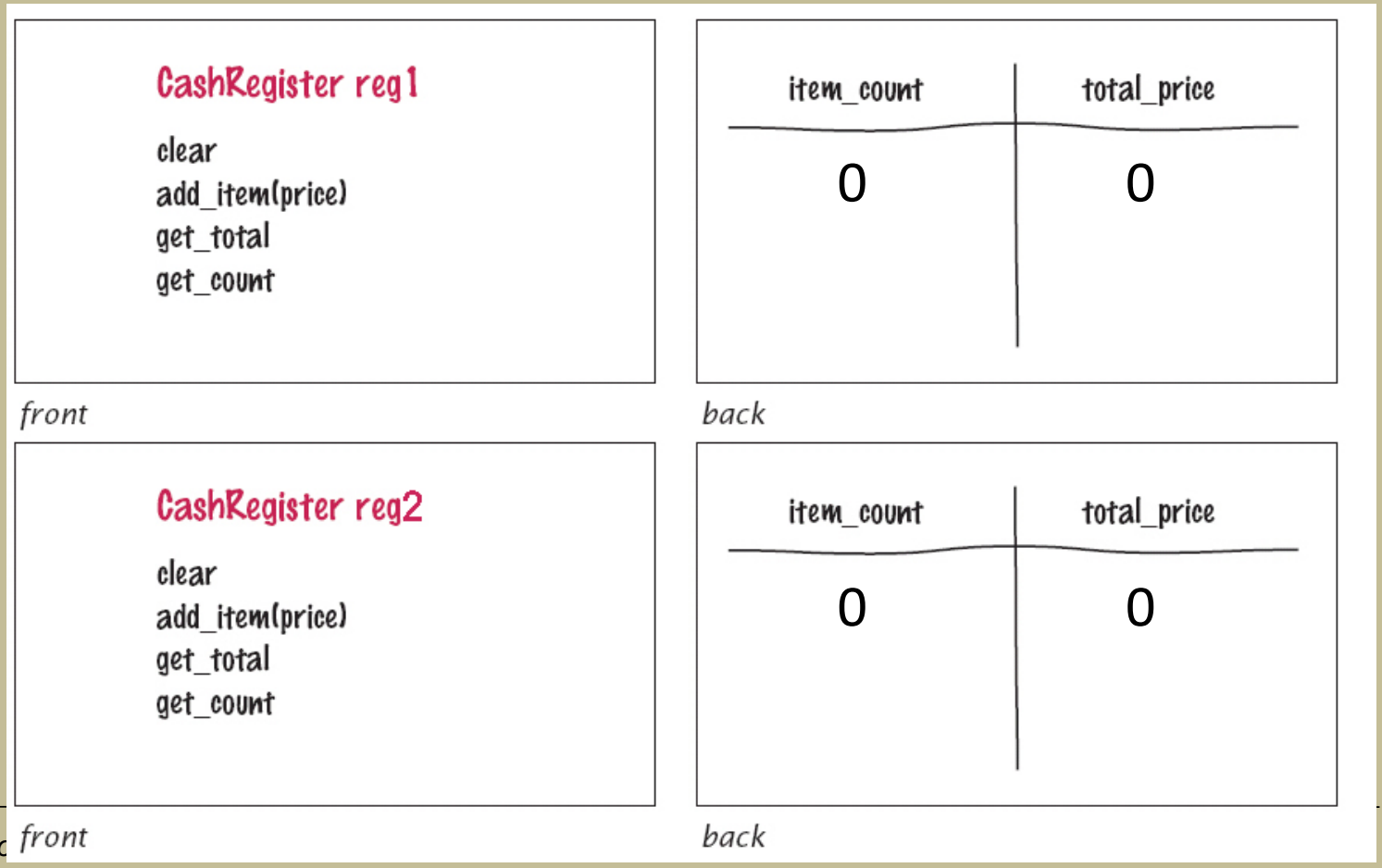
When an object is constructed, add the variable's name to the front of a card and fill in the initial values.



Tracing Objects

```
CashRegister reg1;
CashRegister reg2;
```

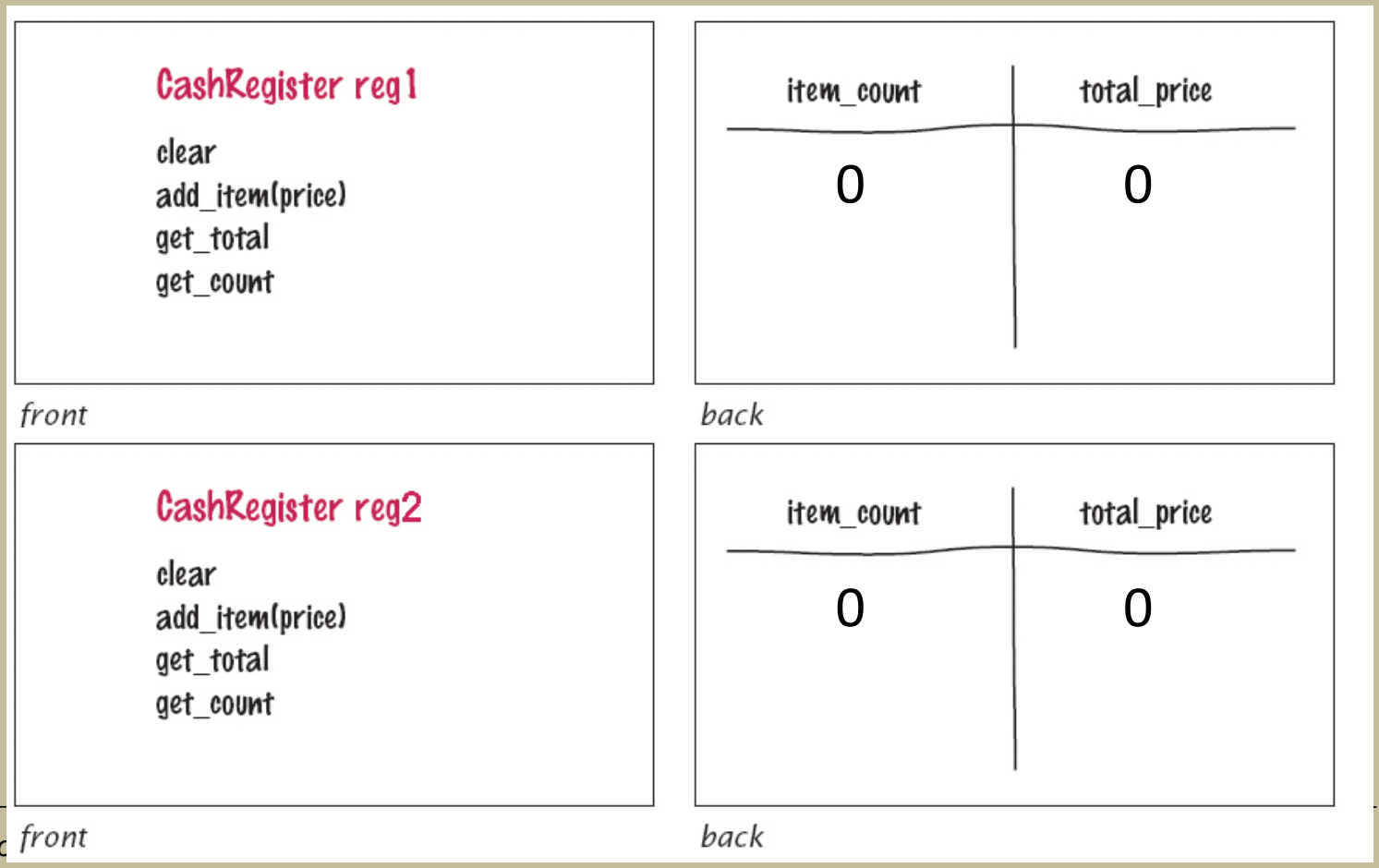
You would do this for every variable.



Tracing Objects

```
CashRegister reg1;
CashRegister reg2;
reg1.addItem(19.95);
```

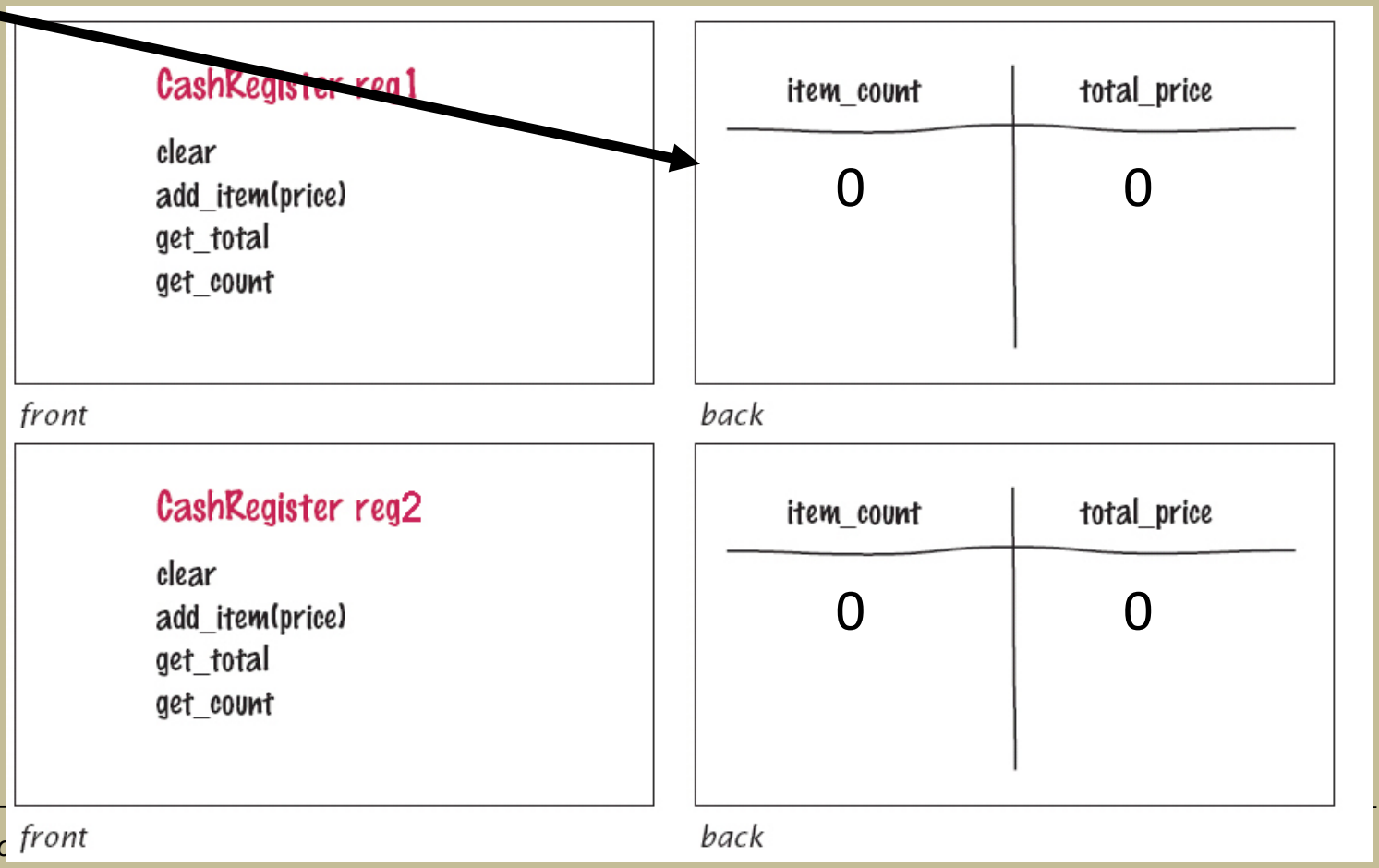
When a method is invoked, grab the right card...



Tracing Objects

```
CashRegister reg1;
CashRegister reg2;
reg1.addItem(19.95);
```

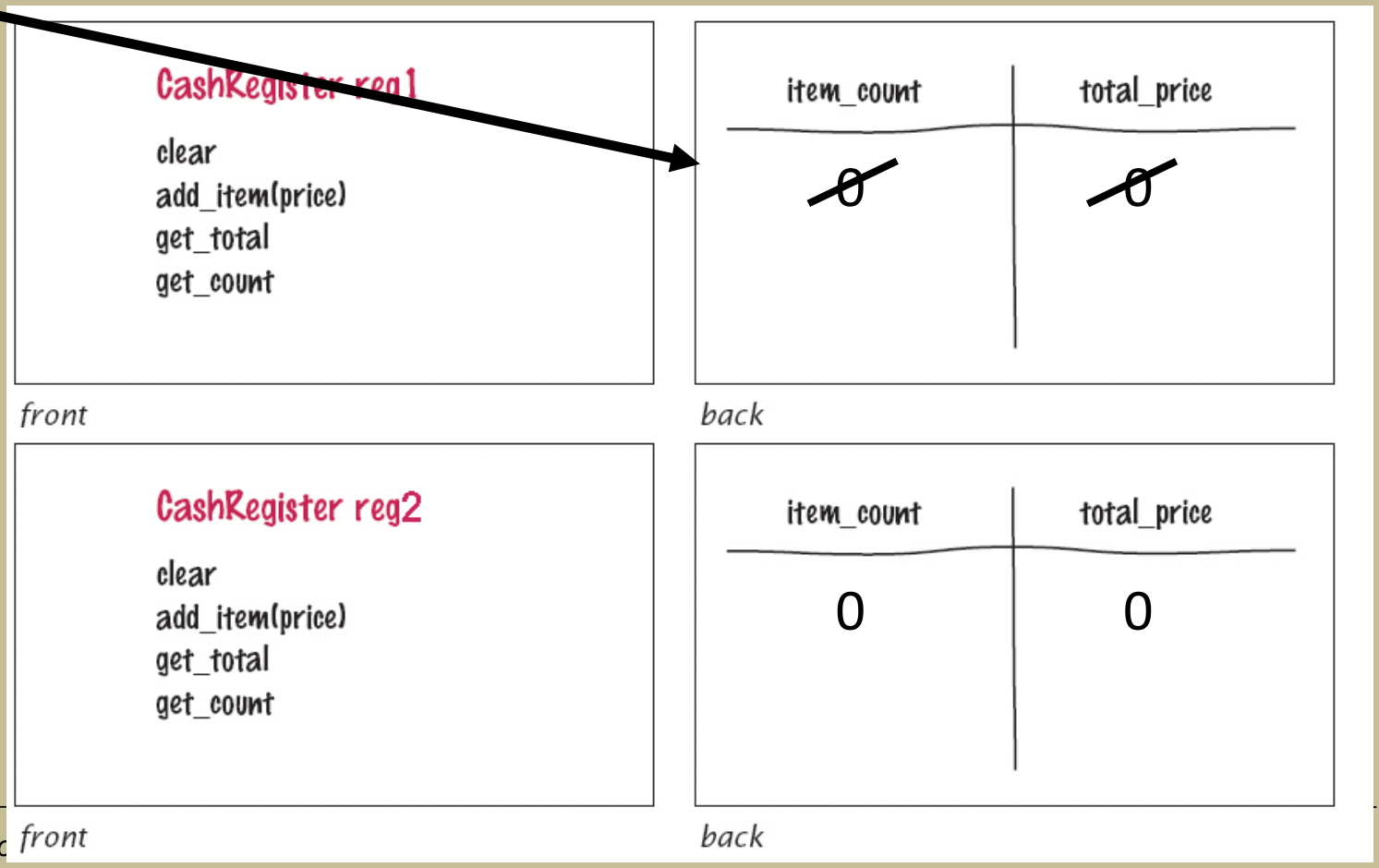
…flip it over…



Tracing Objects

```
CashRegister reg1;
CashRegister reg2;
reg1.addItem(19.95);
```

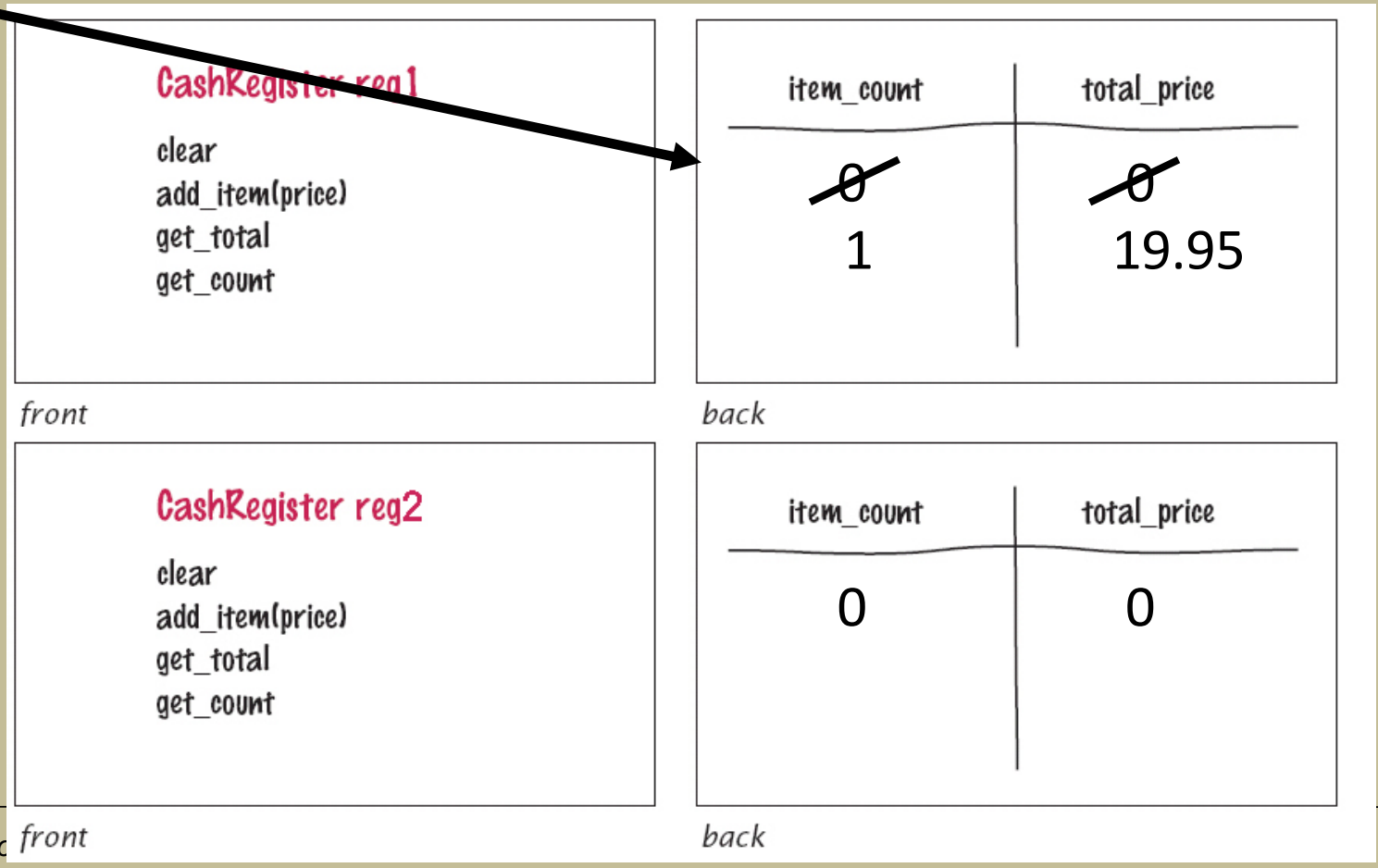
…cross out the old values...



Tracing Objects

```
CashRegister reg1;
CashRegister reg2;
reg1.addItem(19.95);
```

…then write the new values below.



Tracing Objects

These cards can help you in development when you need to add more functionality:

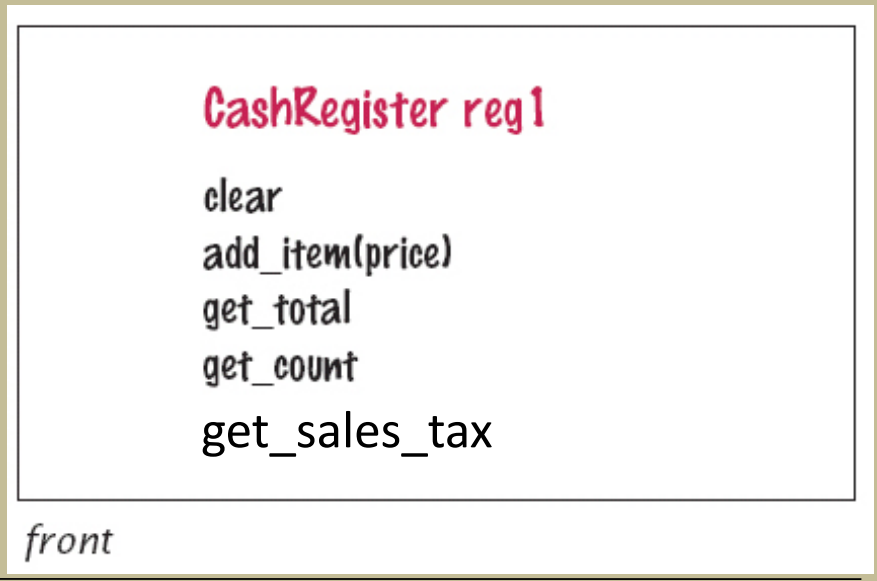
Suppose you are asked to get the sales tax.

Tracing Objects

You would add that to the front of the cards.
Grab any card – they will all have to be redone.

Add the newly requested method.

Then flip it over and start thinking.

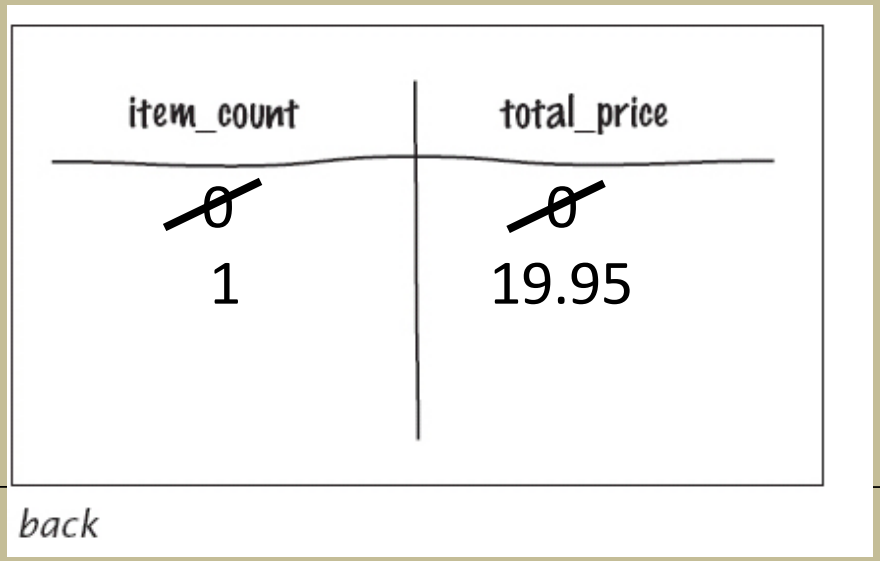


Tracing Objects

You would add that to the front of the cards.
Grab any card – they will all have to be redone.

Add the newly requested method.

Then flip it over and start thinking.

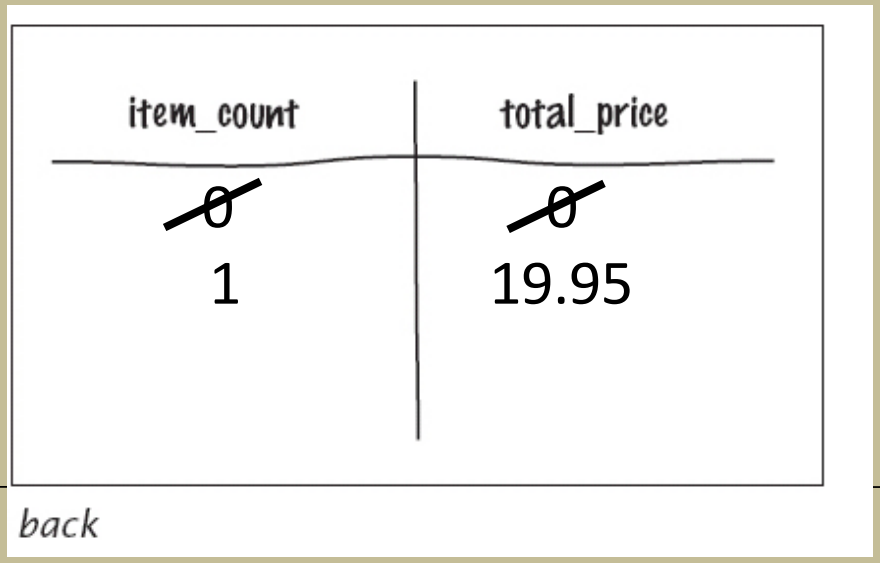


Tracing Objects

I have to calculate the sales tax.

Do I have enough information here on the back of this card?

I can only use these and any values passed in through parameters and global variables.



Tracing Objects

Tax rate?

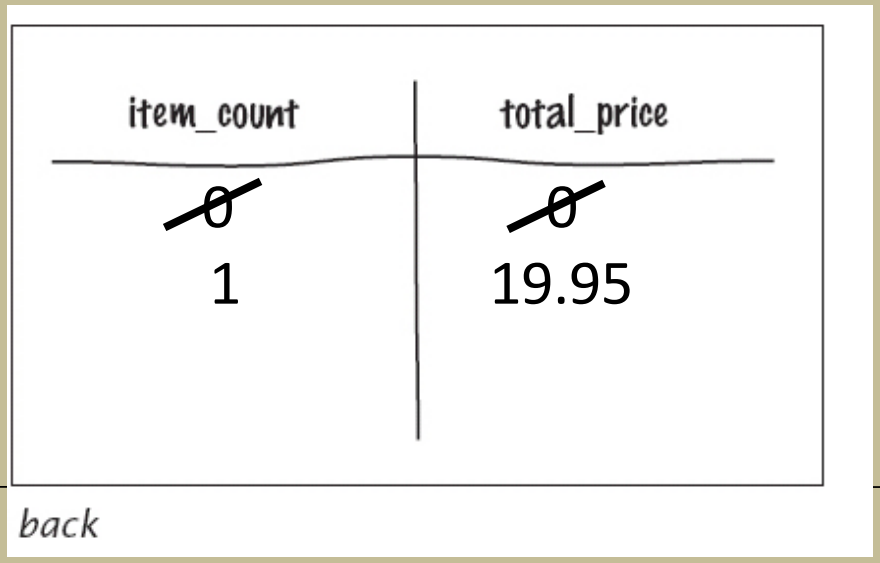
Need a new data member **`tax_rate`** for this which would be set in the constructor to a global constant.

Are all items taxable?

Need to add another parameter for taxable-or-not to **`add_item`** which would appropriately update...

…what???

Need a new data member: **`taxable_total`**.



Tracing Objects

Add these things and do some tracing.

```
CashRegister reg2(TAX_RATE);
reg2.addItem(3.95, false);
reg2.addItem(19.95, true);
```

item_count	total_price	taxable_total	tax_rate
~~0~~	~~0~~	~~0~~	7.5
~~1~~	~~3.95~~		
2	23.90	19.95	

Example

c9– account.cpp

Discovering Classes

Often times,

- nouns correspond to classes, and
- verbs correspond to member functions.

Discovering Classes

Many classes are abstractions of real-life entities.

- **`BankAccount`**
- **`CashRegister`**

Discovering Classes

Generally, concepts from the problem domain,
be it science, business, or a game, make good classes.

The name for such a class should be
a noun that describes the concept.

Other frequently used classes represent
system services such as files or menus.

Not Discovering Classes

What might *not* be a good class?

If you can't tell from the class name what an object of the class is supposed to do, then you are probably not on the right track.

Not Discovering Classes

For example, you might be asked to write a program that prints paychecks.

You start by trying to design a **`class PaycheckProgram`**.

Not Discovering Classes

```
class PaycheckProgram
```

?

What would an object of this class do?

Not Discovering Classes

```
class PaycheckProgram
```

? ?

An object of this class would have to

do everything!

Not Discovering Classes

```
class PaycheckProgram
```

? ? ?

That doesn't simplify anything.

A better class would be:

Discovering Classes

```
class Paycheck
```

! ! ! ! !

Not Discovering Classes

Another common mistake, made particularly by those who are used to writing programs that consist of functions, is to turn an *action* into a *class*.

Not Discovering Classes

For example, if you are to compute a paycheck, you might consider writing a

`class ComputePaycheck`.

Not Discovering Classes

`class ComputePaycheck`

But can you visualize a "**`ComputePaycheck`**" object?

A *thing* that *is* a **`computePaycheck`**?

Not Discovering Classes

```
class ComputePaycheck
```

The fact that "computepaycheck" is ***not a noun***
tips you off that you are on the wrong track.

On the other hand, a "paycheck" class makes intuitive sense.

(The word "paycheck" is a noun.)

Discovering Classes

You can visualize a paycheck object.

You can then think about useful member functions of the **`Paycheck`** class, such as **`compute_taxes`**, that help you solve the problem.

“Has-a” relationship

When you analyze a problem description,
you often find that you have multiple classes.

It is then helpful to consider how these classes are related.

One of the fundamental relationships between classes
is the “*aggregation*” relationship

(which is informally known as the “has-a” relationship).

“Has-a” relationship

The aggregation relationship states that objects of one class contain objects of another class.

“Has-a” relationship

Consider a quiz that is made up of questions.

Since each quiz has one or more questions,
we say that the class **`Quiz`** *aggregates* the class **`Question`**