Get the order right!

```
public Account(String newName, int newNumber){
    name = newName;
    number = newNumber;
}

public Account(String newName, int newNumber,
           double initialBalance){
    this(newName,newNumber);
    balance = initialBalance;


}
```

`this` keyword

- `this(`*parameter list*`)` refers to another constructor in the class you are currently writing in
- *in* this *object, find a constructor that has the appropriate parameter list*
- `this` is referring to the *implicit* parameter

Using `this` with shadowing

```
public class Account{
    String name;
    int number;
    ...
    public void setNumber(int number) {
        this.number = number;

    }
}
```

same value

same value

- Inside the `setNumber` method,
  - When `this.number` is seen, the *instance variable* `number` is used.
  - When `number` is seen, the *parameter* `number` is used.

Or avoid shadowing altogether

```
public class Account{
    String name;
    int number;

    ...
    public void setNumber(int inputNumber) {
        number = inputNumber;

    }
}
```

```
public class Account{
   ...
   public void updateBalance(){
       int increase = this.calcInterest();
       balance = balance + increase;
   }
   ...
   private int calcInterest(){
       code to generate monthly interest on current
   }
```

a mutator method that adds monthly interest to an account balance

calls another method within the class; here `this` is optional but makes the code more clear to the reader

notice this method is `private`
this particular helper method is only used within the class

`this` keyword

- **`this`** : Refers to the implicit parameter.
  - *implicit parameter*: the object on which a method is called

- Common uses for `this`:
  - To call one constructor from another constructor:
    `this``(parameters);`
  - To refer to an instance variable (often optional):
    `this``.field`
  - To call another method in the class (often optional):
    `this``.method(parameters);`

Two useful methods to include in your class

- `toString`
- `equals`

✓ in the AccountClientProgram.java, add code to print acct1

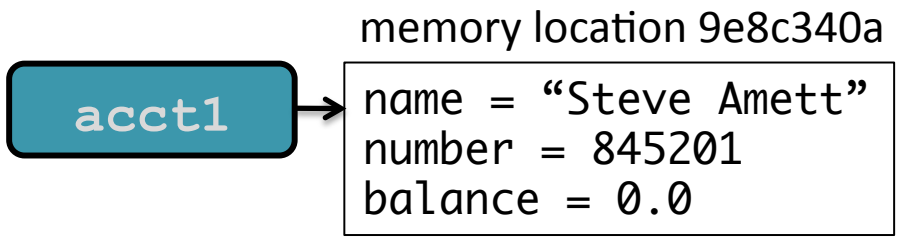
```
out.println(acct1);
```

✓what is that output?

Printing objects

- We are using the default definition of `toString()`:

```
Account acct8 = new Account("Hi H. Silver", 389024);
out.println(acct8);
```
prints `Account@9e8c340a`



Printing objects

- We are using the default definition of `toString()`:

```
Account acct8 = new Account("Hi H. Silver", 389024);
out.println(acct8);
```
prints `Account@9e8c340a`

- We can print a more useful string (but this is cumbersome):

```
out.println("acct8: " + acct8.getName() + ", " +
  acct8.getNumber());
```

- *Override* the default `toString` method

Override the default `toString` method

```
public String toString() {
    code that returns a suitable String ;
}
```

keyword `this` is optional but may help with readability

- The method name, return type, and parameters must match exactly.

```
public String toString() {
        return ("Account holder:  + this.getName() + ", " + getNumber());
}
```

Back in the client program…

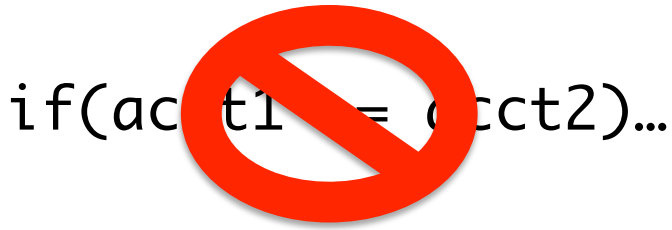
```
out.println(acct8.toString());
```

Output:

```
Account holder: Hi H. Silver, 389024
```

Two useful methods to include in your class

- `toString`
- `equals`



Method `equals`

- What does it mean for two objects to be equal?
  - two objects may be **equal** when the values of only one particular instance variable match.
  - two objects may be **equal** only when the values of all instance variables match.
  - two objects may be **equal** if the instance variables are within a certain range of each other

- <u>Always</u> name the method `equals`.

Method `equals`

object.`equals`(*another object*)

- ✓ Check to see if the other object is the same type of object
- ✓ If it is
  - ✓ Make necessary comparisons to determine equality
- ✓ else return false
  - ✓ An `Account` object can never be equal to a `chessboard` object for example

Method `equals`

```
public boolean equals (Object other) {
    if (other instanceof Account) {
        Account otheracct = (Account) other;
```

We cannot access any fields of `otherAcct` until the object is cast as an Account object.

We can access `otherAcct.number` even though it is private because it is an Account object and this is a method of the Account class

```
}
```

A class file should contain

- Constructors
  - use the default constructor OR
  - write constructors of your own
  - no return type, same name as class, overloading
- Accessors
  - make instance variables private
  - use accessors like `getName()` to access values

A class file should contain

- Mutators
  - use simple set methods like `setName(x)` to overwrite instance variable values
  - use more complex methods like `withdrawal(x)` to change values
- the `toString` method
  - to define how an object's data should be displayed
  - the default is not very useful
  - method is overridden when defined in the class
  - can be called implicitly `out.print(acct3);`

A class file should contain

- the equals method
  - to compare two objects
  - method is overridden when defined in the class
  - if (acct1.equals(acct7)…