A Bank Account Example

- Essentially based on Big Java
 - But using explicit this references
 - And putting fields at the top of the class
- Comparing and contrasting with Python
 - Source code with Python examples is in SVN for reference
- Next slide shows the entire class
 - Subsequent slides discuss it piece by piece

The BankAccount class

290

30

31 32

33

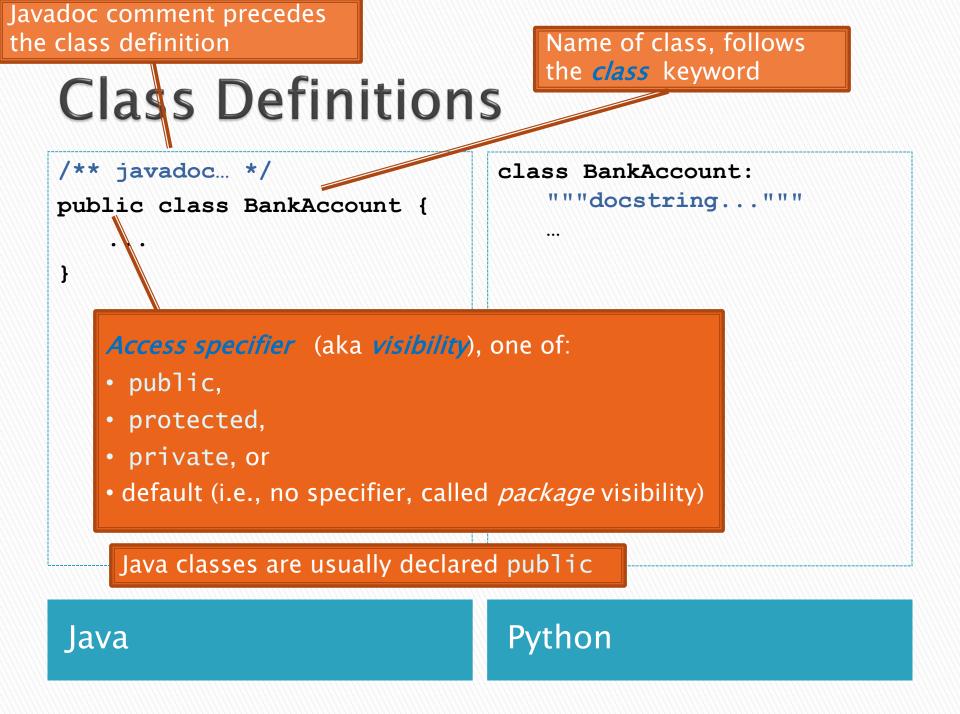
34

350

```
36
 10/**
                                                  37
 2
     * A BankAccount has a balance that can be
                                                  38
 3
     * changed by deposits and withdrawals.
                                                  39
 4
     *
                                                  400
 5
     * @author Cay Horstmann.
                                                  41
 6
     */
                                                  42
 7
   public class BankAccount {
                                                  43
 8
                                                  44
        private double balance;
                                                  45
 9
                                                  460
100
        / * *
                                                  47
11
         * Constructs a bank account
                                                  48
12
         * with a zero balance.
                                                  49
13
         */
                                                  50
140
        public BankAccount() {
                                                  510
15
            this.balance = 0.0;
                                                  52
16
        }
                                                  53
17
                                                  54
                                                  55
180
        /**
                                                  560
19
         * Constructs a bank account with a
                                                  57
20
           given initial balance.
         *
                                                  58
21
         *
                                                  59 }
           @param initialBalance
22
         π.
23
         *
                       the initial balance
24
         *7
250
        public BankAccount(double initialBalance) {
26
            this.balance = initialBalance;
27
```

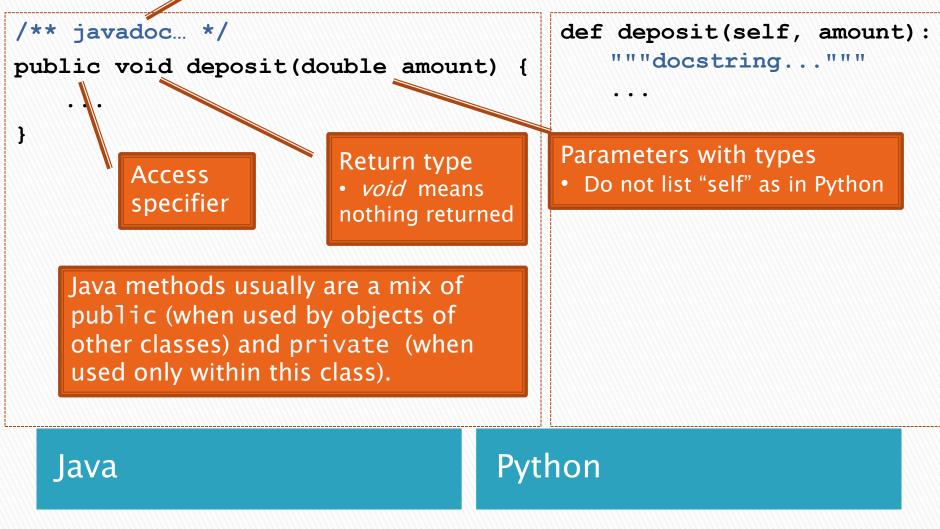
```
/ * *
 * Deposits money into the bank account.
 *
 * @param amount
 *
              the amount to deposit
 */
public void deposit(double amount) {
    double newBalance = this.balance + amount;
    this.balance = newBalance;
}
/ * *
 * Withdraws money from the bank account.
 *
   Oparam amount
 *
               the amount to withdraw
 *7
public void withdraw(double amount) {
    double newBalance = this.balance - amount;
    this.balance = newBalance;
}
/**
 * Returns the current balance.
 * @return the current balance
 */
public double getBalance() {
    return this.balance:
}
```

A class has 3 parts after its header: *fields*, *constructors* and *methods*.



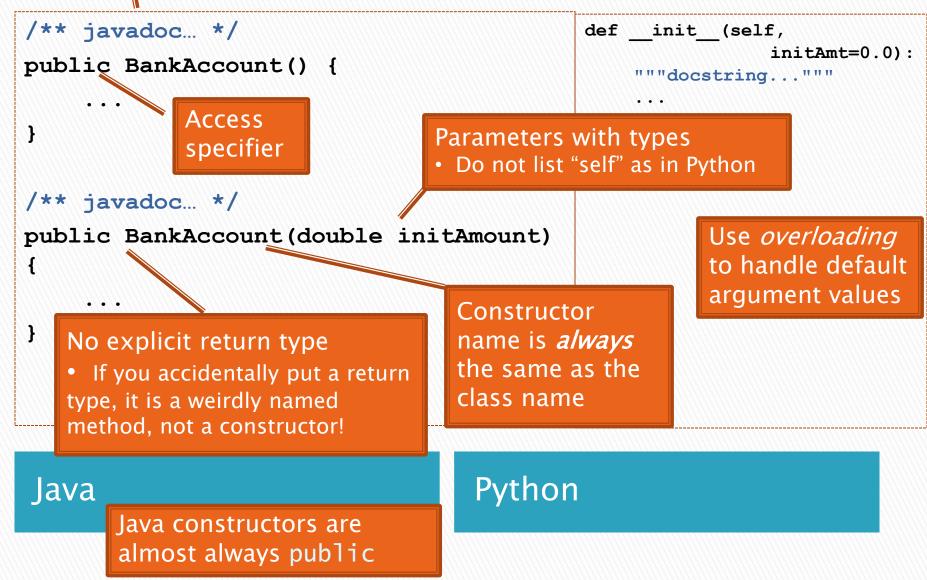
Javadoc comment precedes the method definition (always if the method is public, optionally if the method is private)





Javadoc comment precedes the constructor definition

Constructor Definitions



Public Interface

- The *public interface* of an object:
 - Is the inputs and outputs of the black box
 - Defines how we access the object as a user
 - Consists of:
 - public constructors of its class, plus
 - public methods of its class
- The *private implementation* of an object consists of:
 - Its (private) instance fields
 - Definitions of its constructors and methods

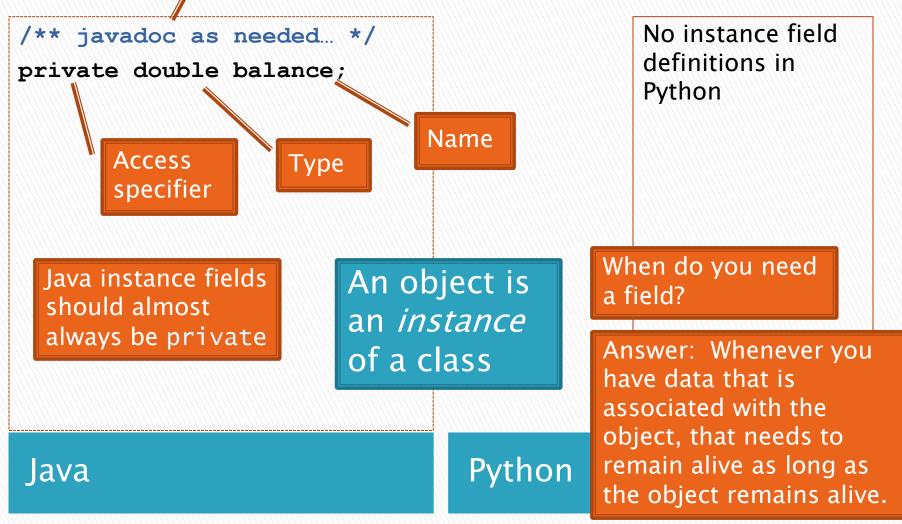
The above shows the public interface of BankAccount objects. The next slides show their private implementation.

BankAccount

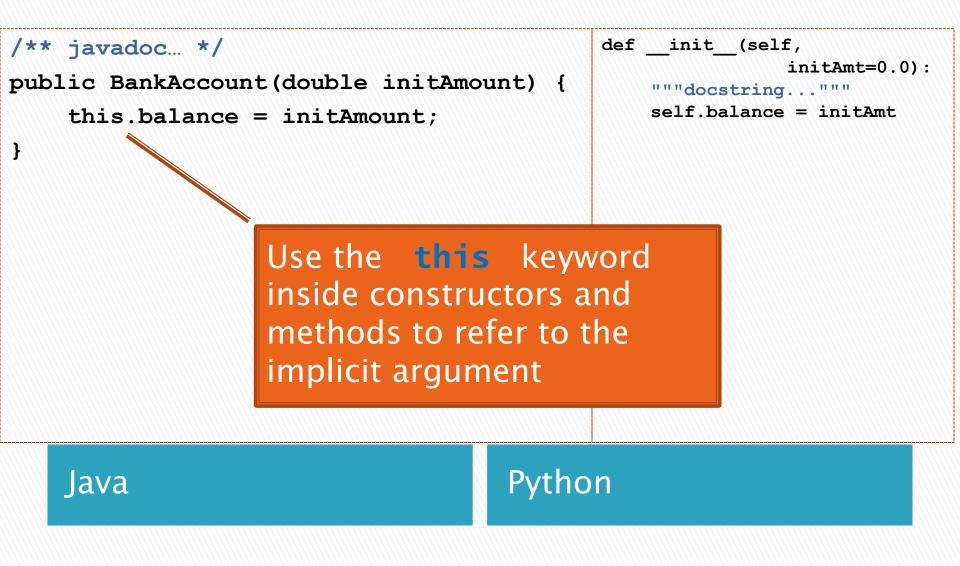
```
BankAccount()
BankAccount(double initAmount)
void deposit(double amount)
void withdraw(double amount)
double getBalance()
```

Generally no Javadoc here, since you should choose variable names that are self-documenting.

Instance Field Definitions



Constructor Implementation

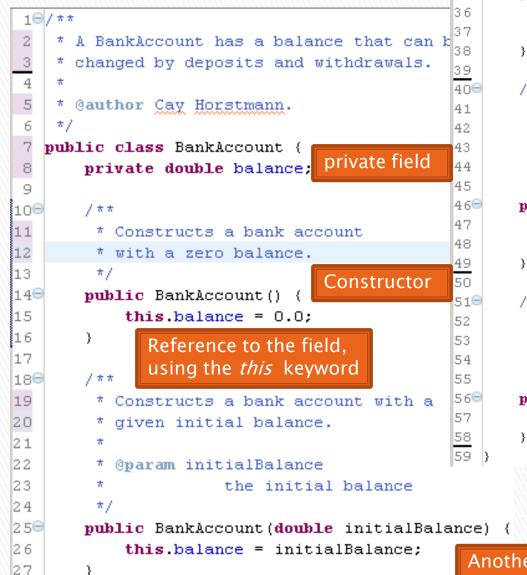


Method Implementation

 Final and a second provide the second provide the second provided and the second provided and	
<pre>/** javadoc */ public void deposit(double amount) { double newBalance = this.balance + amount; this.balance = newBalance;</pre>	<pre>def deposit(self, amount): """docstring"""" newBal = self.balance + amount self.balance = newBal</pre>
<pre>/** javadoc */ public double getBalance() { return this.balance; }</pre>	def getBalance(self): """docstring""" return self.balance

Java Can omit *return* for *void* methods Python

The *BankAccount* class (summary)



290

30

31 32

33

34 359

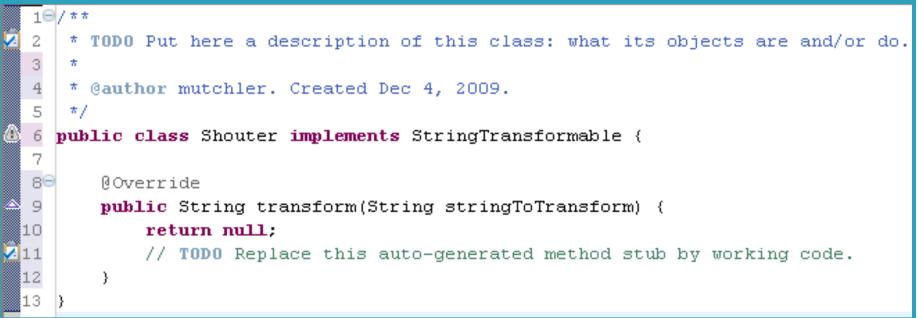
```
/ * *
 * Deposits money into the bank account.
 * @param amount
 *
              the amount to deposit
 *7
public void deposit(double amount) {
    double newBalance = this.balance + amount;
    this.balance = newBalance;
}
      deposit method. Note the use of a
      parameter, local variable and field.
/ * *
 * Withdraws money from the bank account.
 π.
  Oparam amount
              the amount to withdraw
 */
public void withdraw(double amount) {
    double newBalance = this.balance - amount;
    this.balance = newBalance:
}
                            Withdraw method
/**
 * Returns the current balance.
 * @return the current balance
 */
public double getBalance() {
    return this.balance:
           A getter method that
           preserves the encapsulation
           of the private field.
```

Another constructor. Note overloading.

Live Coding

WordGames Shouter

Shouter – After Eclipse writes stubs for you



Step 1: Create the (initially empty) class

• File \Rightarrow New \Rightarrow Class

Step 2: Write *documented stubs* for the *public interface* of the class

Do you understand what it means to *implement an interface*? Do you see what a *stub* is? Did you see how Eclipse offered to write the stubs for you? Note the TODO's: The above is not yet a *documented* stub – see the next slide for that.

Shouter – After you DOCUMENT your stubs

```
19/**
 2
    * A Shouter "shouts". That is, given blah, it produces the result of changing
 3
    * all the characters in blah to upper-case.
                                                                 Do you see the form for Javadoc
 4
                                                                 comments? For their tags?
 5
    * @author David Mutchler. Created December 4, 2009.
 6
    */
   public class Shouter implements StringTransformable {
 7
 8
                                                                 The form for a class?
 90
       /**
10
         * "Shouts". That is, given blah, returns the result of changing all the
11
        * characters in blah to upper-case.
12
         *
        * @param stringToTransform
13
14
        * @return the result of changing all the characters in the given String to
15
         *
                   upper-case.
16
        */
170
       00verride
18
       public String transform(String stringToTransform) {
19
           return null:
20
           // TODO Replace this auto-generated method stub by working code.
21
       - }
22
```

Step 1: Create the (initially empty) class

• File \Rightarrow New \Rightarrow Class

Step 2: Write *documented stubs* for the *public interface* of the class

Do you understand what it means to use *documented stubs*? Do you know what you must document? (Answer: anything *public*.)

ShouterTest

```
implementing?
12
   public class ShouterTest {
13
       private Shouter shouter;
                                                             Do you see what a field is?
14
                                                             Why one is used here? (Answer:
15\Theta
       / * *
16
        * Runs before each test, constructing for each test
                                                             so the Shouter can be reused in all the
17
         *
                                                             tests. It would also be OK to construct a
18
        * @throws java.lang.Exception
                                                             new Shouter for each test.)
19
        */
200
       @Before
                                                             Did you see how the
21
       public void setUp() throws Exception {
                                                             assertEquals method works?
22
           this.shouter = new Shouter();
23
       }
                                                             How you specify a test? How the
24
                                                             @Before and @Test annotations work?
250
       /**
26
        * Test method for {@link Shouter#transform(java.lang.String)}. Tests that a
27
        * string in all upper case stays that way.
                                                                        Look at the (many)
28
        *7
290
       RTest
                                                                        tests we supplied in
30
       public void testAllUpperCase() {
                                                                         ShouterTest. Are they
31
           String upperCase = "CAPS LOCK IS CRUISE CONTROL";
32
                                                                         a good set of tests, with
33
           assertEquals(upperCase, this.shouter.transform(upperCase));
                                                                         good coverage? Could
34
       }
                                                                         we test how fast Shouter's
          Step 1: Create the (initially empty) class
                                                                         transform runs?
```

Do you understand why you

write tests before

Step 2: Write *documented stubs* for the *public interface* of the class

Step 3a: We provided *some JUnit tests* for the *transform* method of each class.

Shouter – After you implement it

```
18/**
     * A Shouter "shouts". That is, given blah, it produces the result of changing
 2
     * all the characters in blah to upper-case.
 3
 4
 5
     * @author David Mutchler. Created December 4, 2009.
 6
     */
   public class Shouter implements StringTransformable {
 8
 90
        / * *
10
         * "Shouts". That is, given blah, returns the result of changing all the
11
         * characters in blah to upper-case.
12
13
         * @param stringToTransform
         * @return the result of changing all the characters in the given String to
14
15
         *
                   upper-case.
16
         */
170
        @Override
18
       public String transform(String stringToTransform) {
19
            return stringToTransform.toUpperCase();
20
        }
21
```

Do you understand how Eclipse helps you find the right method to apply to the *stringToTransform*? (Pause after typing the dot.)

Do you see why you don't need a local variable?

Do you know Java's 1st dirty little secret about constructors? (Namely, that Java inserted a do-nothing constructor for you! More on this later.)

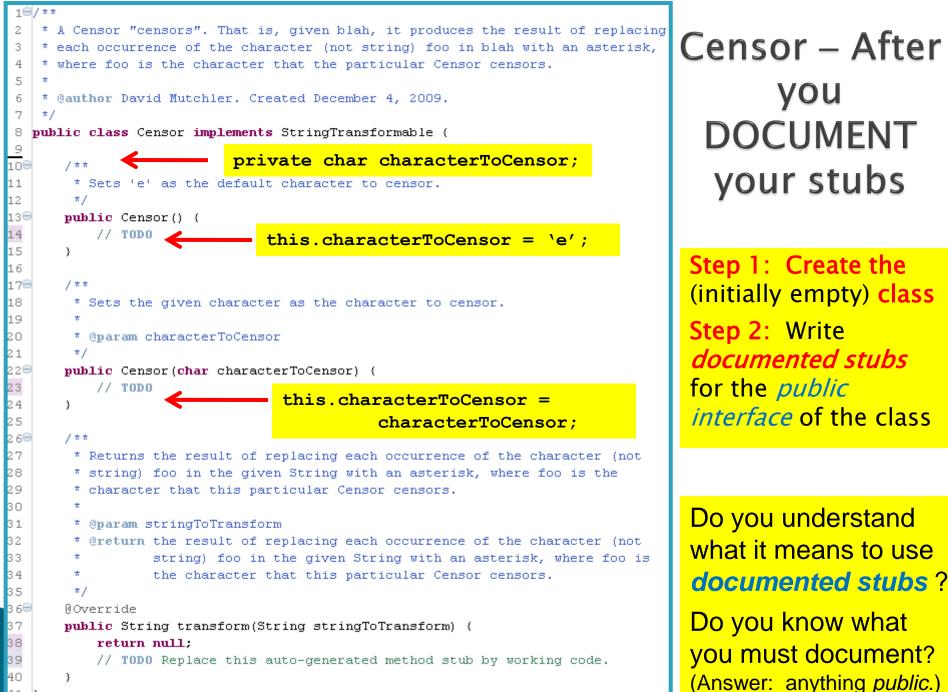
Censor

- Censor: given a string *inputString*, produces a new string by replacing each occurrence of charToCensor with a "*" (an asterisk).
- How do you deal with charToCensor ?
 - Can it be a parameter of *transform*?
 - No, that violates the StringTransformable interface
 - Can it be a local variable of *transform*?
 - No, it needs to live for the entire lifetime of the Censor.
 - What's left?
 - Answer: It is a *field* ! (What is a sensible name for the field?)
- How do you initialize the field for **charToCensor**?
 - Answer: by using Censor's constructors!

Live Coding

WordGames Censor

Censor – After Eclipse writes stubs for you 10/** * TODO Put here a description of this class: wha Step 1: Create the (initially empty) class 2 3 * Step 2: Write *documented stubs* for * @author mutchler. Created Dec 7, 2009. 4 the *public interface* of the class */ 5 public class Censor implements StringTransformable { 7 88 /** 9 * TODO Put here a description of what this constructor does. 10 */ Do you see why you need stubs for 110 public Censor() { 12 // TODO the two Censor constructors? (See 13 } the calls to them in the CensorTest class.) 14 150 / * * * TODO Put here a description of what this constructor does. 16 17 * @param characterToCensor Do you understand what it means to 18 */ implement an interface? 190 public Censor(char characterToCensor) { Do you see what a **stub** is? Did you see how 20 // **TODO** Eclipse offered to write the stubs for you? 21 } Note the TODO's: The above is not yet a 22 **documented stub** – see the next slide for that. 230 **ROverride** 24 public String transform(String stringToTransform) { 25 return null: 26 // TODO Replace this auto-generated method stub by working code. 27 3 28



41 }

```
public class Censor implements StringTransformable {
  8
  9
                                                                  Do you see why Censor
        private char characterToCensor;
 10
 11
                                                                   needs a field? How the
 120
        / * *
         * Sets 'e' as the default character to censor.
 13
                                                                  field is initialized? How
 14
         */
                                                                  the field is referenced
 15 \odot
        public Censor() {
                                                Censor
            this.characterToCensor = 'e';
 16
                                                                  (using this)?
                                            final version
 17
        }
 18
                                                                   How Censor has two
 190
        / * *
 20
         * Sets the given character as the character to censor.
                                                                   constructors? How those
 21
                                                                   constructors are called in
 22
         * @param characterToCensor
 23
         *7
                                                                  CensorTest?
 240
        public Censor(char characterToCensor) {
 25
            this.characterToCensor = characterToCensor;
                                                                   Should we have made a field for the
 26
        3
                                                                  "
    "
    "
    constant? (Probably.)

 27
 280
        / * *
 29
         * Returns the result of replacing each occurrence of the character (not
         * string) foo in the given String with an asterisk, where foo is the
 30
 31
         * character that this particular Censor censors.
 32
         *
 33
         * @param stringToTransform
 34
         * @return the result of replacing each occurrence of the character (not
 35
         *
                   string) foo in the given String with an asterisk, where foo is
 36
         *
                   the character that this particular Censor censors.
 37
         *7
 380
        AOverride
2339
        public String transform(String stringToTransform) {
 40
            return stringToTransform.replace(this.characterToCensor, '*');
 41
        -}-
 42 }
```