Software Development (CS2500)

Lecture 4: Introduction to Objects and Classes (Continued)

M. R. C. van Dongen

September 30, 2013

Introduction

- Let's implement a Class for bank accounts.
- The class should provide the following services:
  - Deposit money into the (current) account;
  - Withdraw money from the (current) account;
  - Get the balance of the (current) account;
- For simplicity all transactions will be in Euro.
- We also assume the account does not have an owner.

Desigining the API

- Knowing the services, we have to decide on the `api`.
- Let's call the class `BankAccount`.
- Assume we have a valid `BankAccount` object.
- Furthemore, assume `current` references the object:
  - `final BankAccount current = new BankAccount( );`
- We'd like to write things such as:
  - `current.deposit( amount );`
  - `current.withdraw( amount );`
  - `final double amount = current.getBalance( );`
- These calls correspond to behaviour:
  - We must implement them as *instance methods*.

Writing the Method Headers

- We've decided on how to use the instance methods.
- We can write a stub implementation for the instance methods.

Java

```
public class BankAccount {
    // instance variable(s)

    public void deposit( final double amount ) {
        // to do
    }

    public void withdraw( final double amount ) {
        // to do
    }

    public double getBalance( ) {
        // to do
    }
}
```

Implementing `getBalance( )`

- It is not completely sure how we should implement the class.
- Let's try and see if we can implement `getBalance( )`.
  - If this goes wrong, we have to try something else.
- The name `getBalance( )` suggests it is a getter method.
  - It suggests it returns the value of an attribute called `balance`.
- `getBalance( )` returns a `double`:
  - This tells us `balance` should also be `double`.

Java

```
public class BankAccount {
    // instance variable(s)

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        // to do
    }
}
```

Implementing `getBalance( )`

- It is not completely sure how we should implement the class.
- Let's try and see if we can implement `getBalance( )`.
  - If this goes wrong, we have to try something else.
- The name `getBalance( )` suggests it is a getter method.
  - It suggests it returns the value of an attribute called `balance`.
- `getBalance( )` returns a `double`:
  - This tells us `balance` should also be `double`.

Java

```
public class BankAccount {
    double balance;

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        // to do
    }
}
```

Implementing `getBalance( )`

- It is not completely sure how we should implement the class.
- Let's try and see if we can implement `getBalance( )`.
  - If this goes wrong, we have to try something else.
- The name `getBalance( )` suggests it is a getter method.
  - It suggests it returns the value of an attribute called `balance`.
- `getBalance( )` returns a `double`:
  - This tells us `balance` should also be `double`.

Java

```
public class BankAccount {
    double balance;

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing the Constructor

- The implementation of `balance` raises a question.
- How do we initiale it?
- We must write a constructor.
- In fact, we must write two:
  - One for new accounts:
    - `new BankAccount( );`
  - One for existing accounts:
    - `new BankAccount( final double amount );`

Implementing the Constructor (Continued)

Java

```
public class BankAccount {

    double balance;

    public BankAccount( ) {
        /* to do */
    }

    public BankAccount( final double amount ) {
        /* to do */
    }

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing the Constructor (Continued)

Java

```
public class BankAccount {

    double balance;

    public BankAccount( ) {
        balance = 0.0;
    }

    public BankAccount( final double amount ) {
        /* to do */
    }

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing the Constructor (Continued)

Java

```
public class BankAccount {
    private static final double INITIAL_BALANCE = 0.0;
    double balance;

    public BankAccount( ) {
        balance = INITIAL_BALANCE;
    }

    public BankAccount( final double amount ) {
        /* to do */
    }

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing the Constructor (Continued)

Java

```
public class BankAccount {
    private static final double INITIAL_BALANCE = 0.0;
    double balance;

    public BankAccount( ) {
        balance = INITIAL_BALANCE;
    }

    public BankAccount( final double amount ) {
        balance = amount;
    }

    public void deposit( final double amount ) { /* to do */ }

    public void withdraw( final double amount ) { /* to do */ }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing `deposit( )` and `withdraw( )`

Java

```
public class BankAccount {
    private static final double INITIAL_BALANCE = 0.0;
    double balance;

    public BankAccount( ) {
        balance = INITIAL_BALANCE;
    }

    public BankAccount( final double amount ) {
        balance = amount;
    }

    public void deposit( final double amount ) {
        /* to do */
    }

    public void withdraw( final double amount ) {
        /* to do */
    }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing `deposit( )` and `withdraw( )`

Java

```
public class BankAccount {
    private static final double INITIAL_BALANCE = 0.0;
    double balance;

    public BankAccount( ) {
        balance = INITIAL_BALANCE;
    }

    public BankAccount( final double amount ) {
        balance = amount;
    }

    public void deposit( final double amount ) {
        balance = balance + amount;
    }

    public void withdraw( final double amount ) {
        /* to do */
    }

    public double getBalance( ) {
        return balance;
    }
}
```

Implementing `deposit( )` and `withdraw( )`

Java

```
public class BankAccount {
    private static final double INITIAL_BALANCE = 0.0;
    double balance;

    public BankAccount( ) {
        balance = INITIAL_BALANCE;
    }

    public BankAccount( final double amount ) {
        balance = amount;
    }

    public void deposit( final double amount ) {
        balance = balance + amount;
    }

    public void withdraw( final double amount ) {
        balance = balance - amount;
    }

    public double getBalance( ) {
        return balance;
    }
}
```

Unit Testing

- *Unit* or component: smallest testable part of the application.
- Having written the `BankAccount` class, we must test its units.
- For `Java` applications, the units are the methods.
- For the moment we do the testing in a separate class.

Testing the Constructor

Java

```
public class TestBankAccount {
    public static void main( String[] args ) {
        // testDefaultConstructor( );
        testSpecialConstructor( );
        // testDeposit( );
        // testWithdraw( );
        // testGetBalance( );
    }

    private static void testSpecialConstructor( ) {
        final double INITIAL_BALANCE = 1.24523243455;
        final BankAccount account = new BankAccount( INITIAL_BALANCE );
        System.err.print( "expected value for initial account is: " );
        System.out.println( INITIAL_BALANCE );
        System.err.print( "actual value for initial account is: " );
        System.out.println( account.getBalance( ) );
    }

    // ...
}
```

Running the Constructor Test

Unix Session

```
$
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
1.24523243455
1.24523243455
$
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
1.24523243455
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
1.24523243455
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u
1.24523243455
$
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
1.24523243455
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u | wc
```

Running the Constructor Test

Unix Session

```
$ javac TestBankAccount.java
$ java TestBankAccount
expected value for initial account is: 1.24523243455
actual value for initial account is:  1.24523243455
$ java TestBankAccount 2> /dev/null
1.24523243455
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u
1.24523243455
$ java TestBankAccount 2> /dev/null | sort -u | wc
      1       1       14
$
```

Object References

- We can only use an object if we have its *reference*.
- The reference is like a remote control.
  - With the remote control, we can make the object do things.
- An object can only do something when one of its instance methods is called *using the object's reference*.

  indirectly Using an object reference variable:
  - `account.deposit( 42.0 )`

  directly Using the object reference:
  - `"Hello world!".length( )`
  - `new BankAccount( ).deposit( 42.0 )`

Object References

- We can only use an object if we have its *reference*.
- The reference is like a remote control.
  - With the remote control, we can make the object do things.
- An object can only do something when one of its instance methods is called *using the object's reference*.

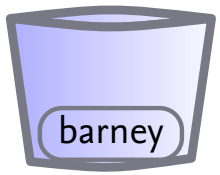
  indirectly Using an object reference variable:
  - `account.deposit( 42.0 )`

  directly Using the object reference:
  - `"Hello world!".length( )`
  - `(new BankAccount( )).deposit( 42.0 )`

Initial Situation

Java

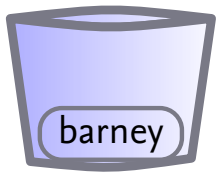
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```



Create new Dog Object

Java

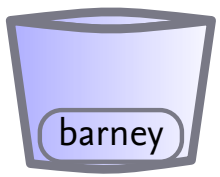
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```

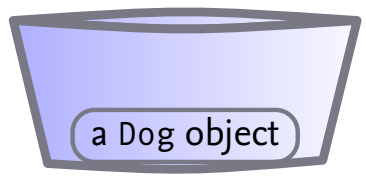


Create new Dog Object: Allocate Space on Heap

Java

```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```

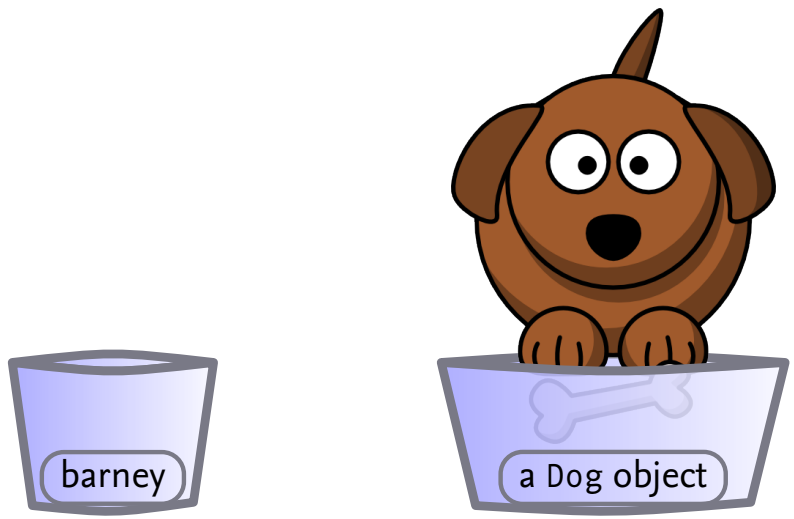




Create new Dog Object: Initialise Dog Object

Java

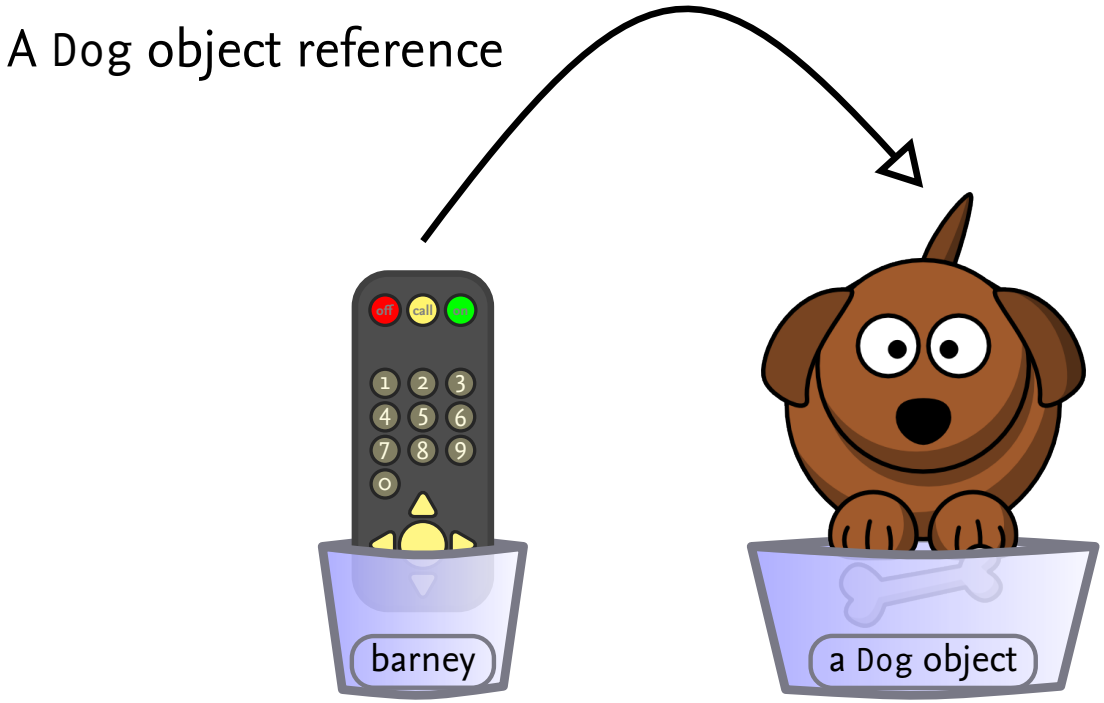
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```



Assign Dog Object Reference to barney

Java

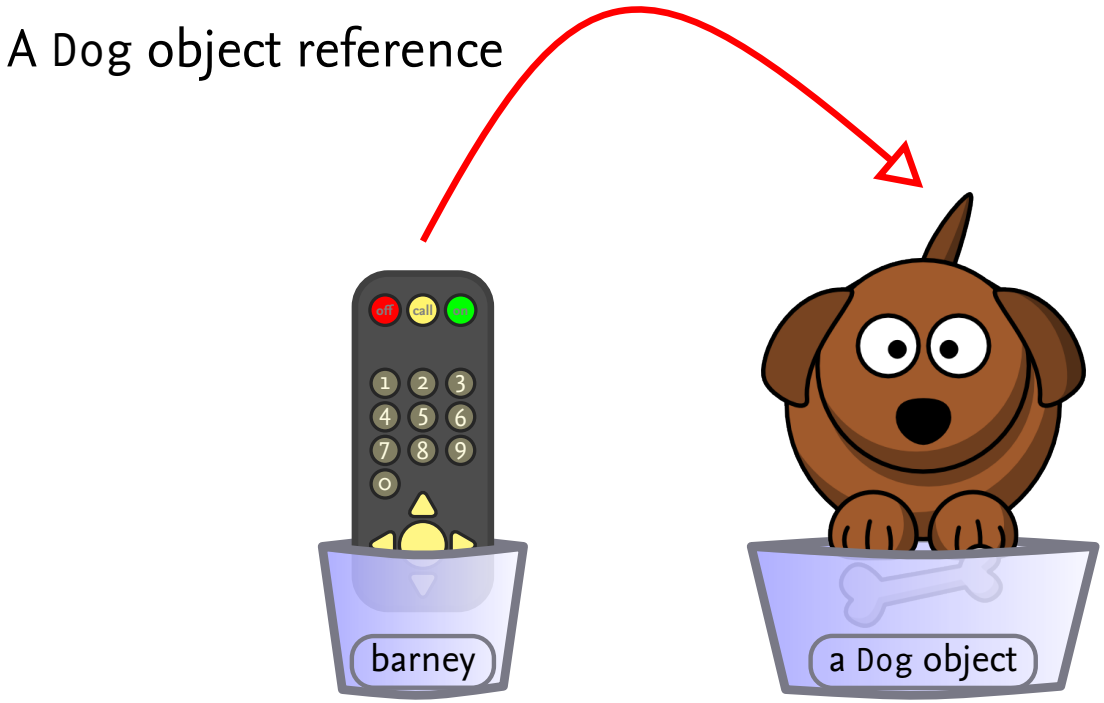
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```



Call barney's Instance Method bark( )

Java

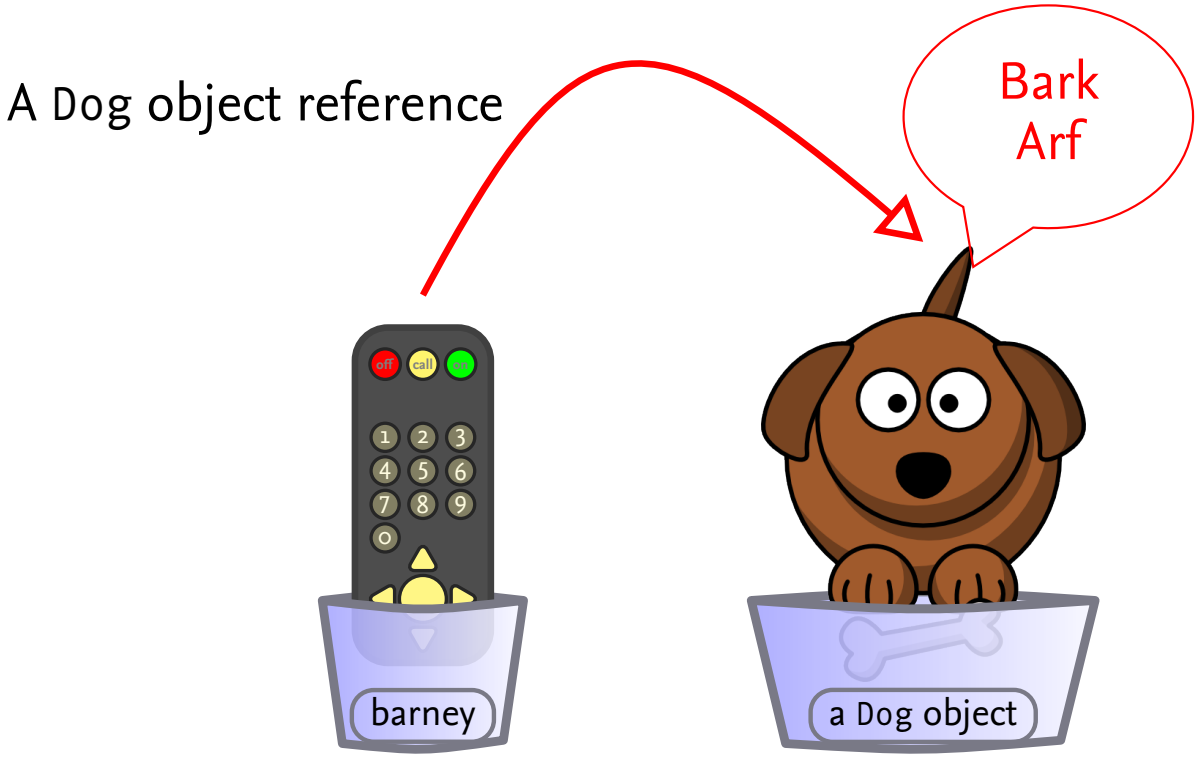
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```



Bark

Java

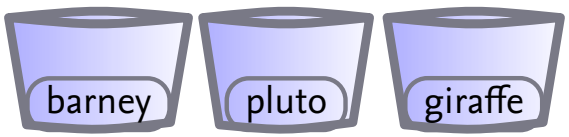
```
// Create new Dog object.
Dog barney = new Dog( );
// Make dog object bark.
barney.bark( );
```



Java Cares about its Types

Java

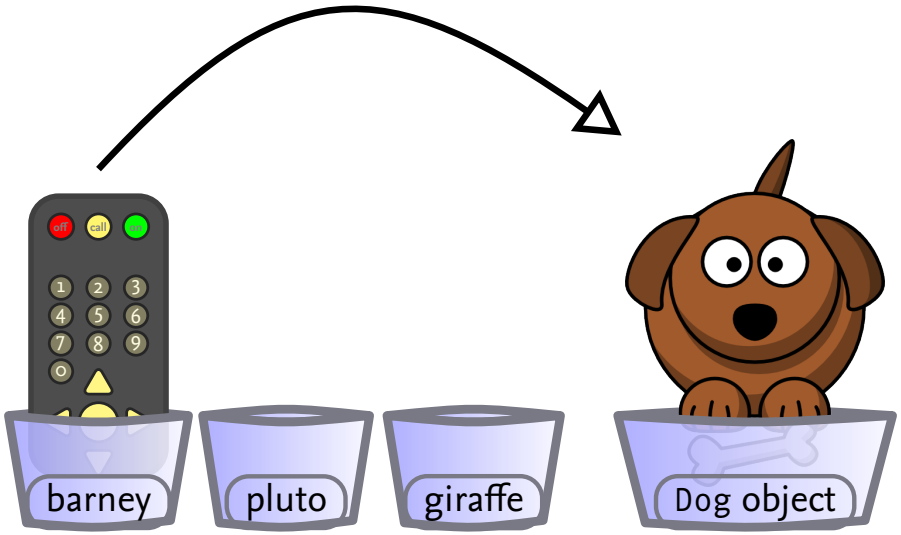
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Giraffe giraffe = new Giraffe( );
```



Java Cares about its Types

Java

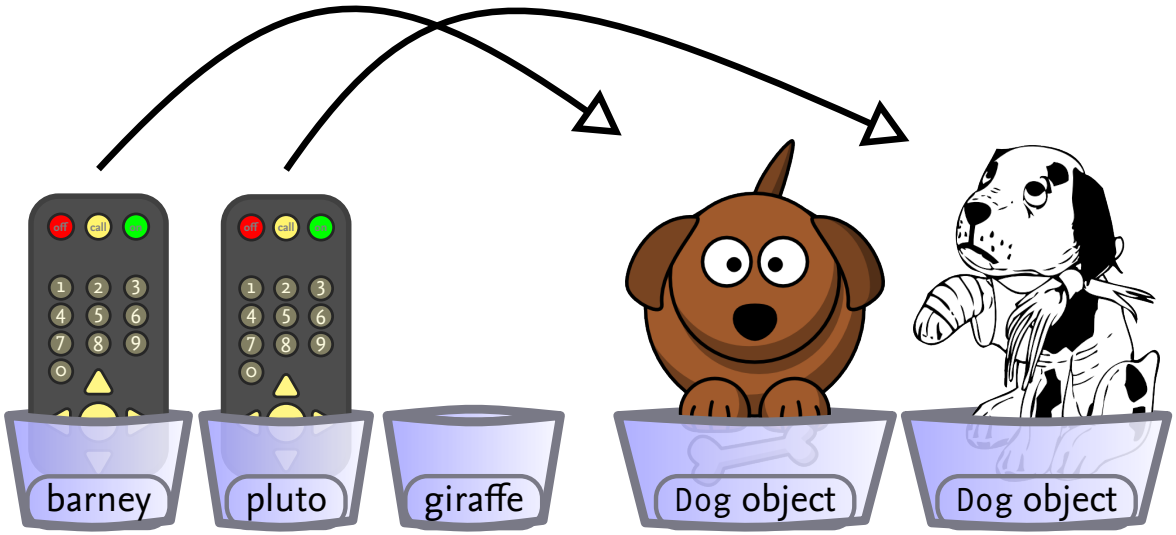
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Giraffe giraffe = new Giraffe( );
```



Java Cares about its Types

Java

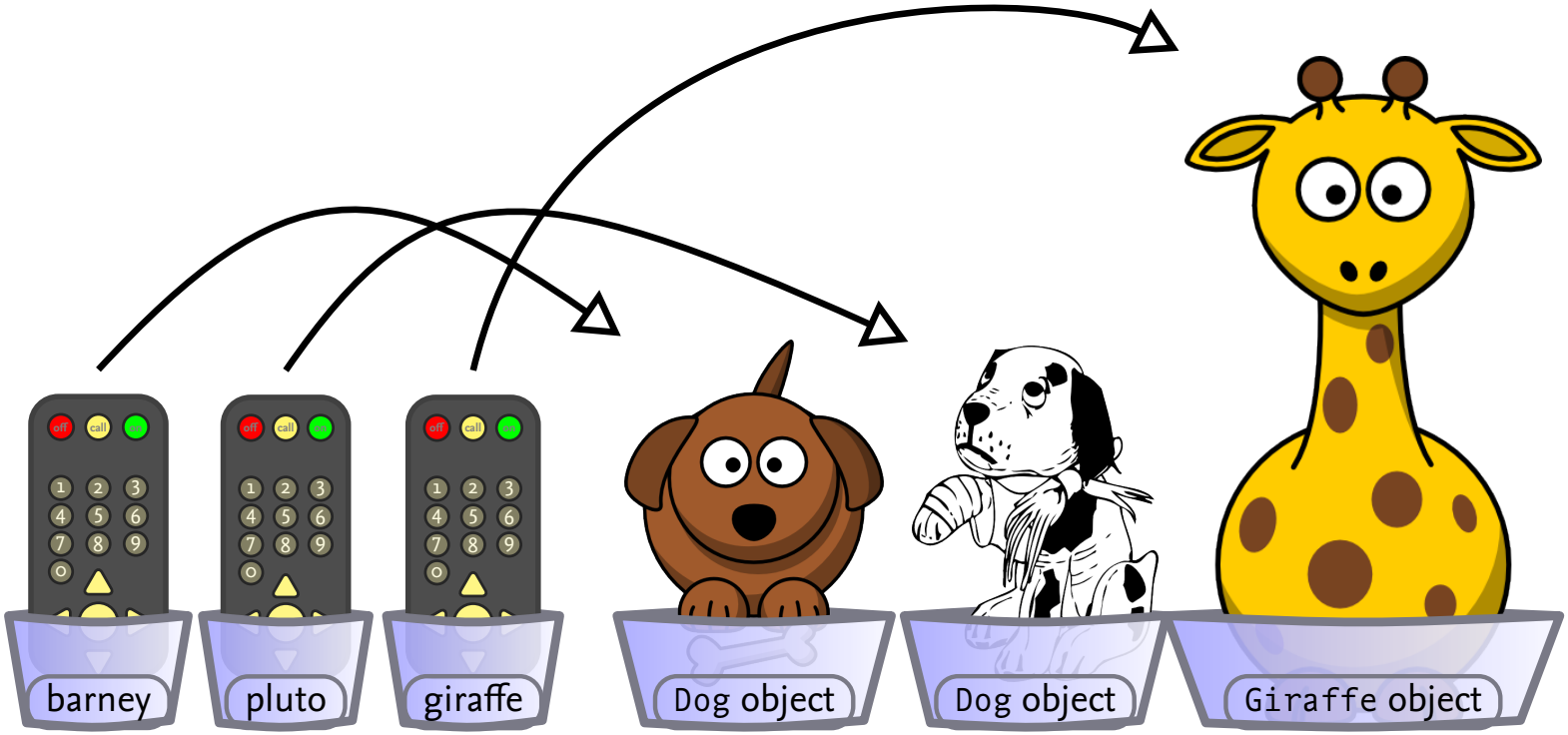
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Giraffe giraffe = new Giraffe( );
```



Java Cares about its Types

Java

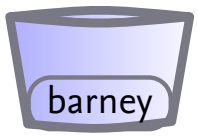
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Giraffe giraffe = new Giraffe( );
```



Types Really Matter

Don't Try This at Home

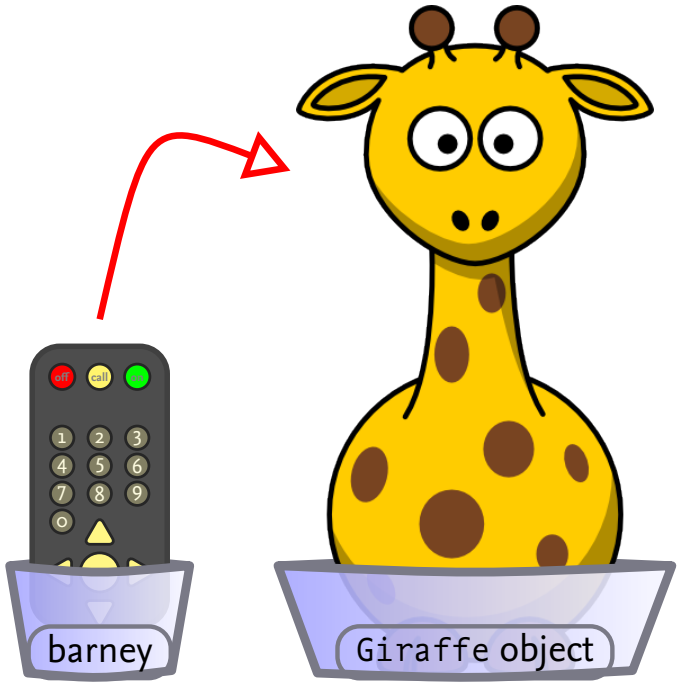
```
Dog barney = new Giraffe( );
```



Types Really Matter

Don't Try This at Home

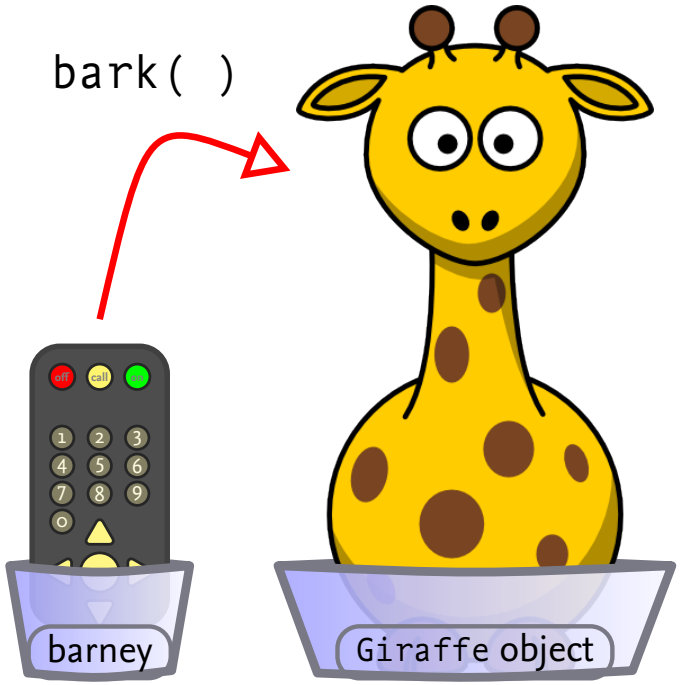
```
Dog barney = new Giraffe( );
```



Types Really Matter

Don't Try This at Home

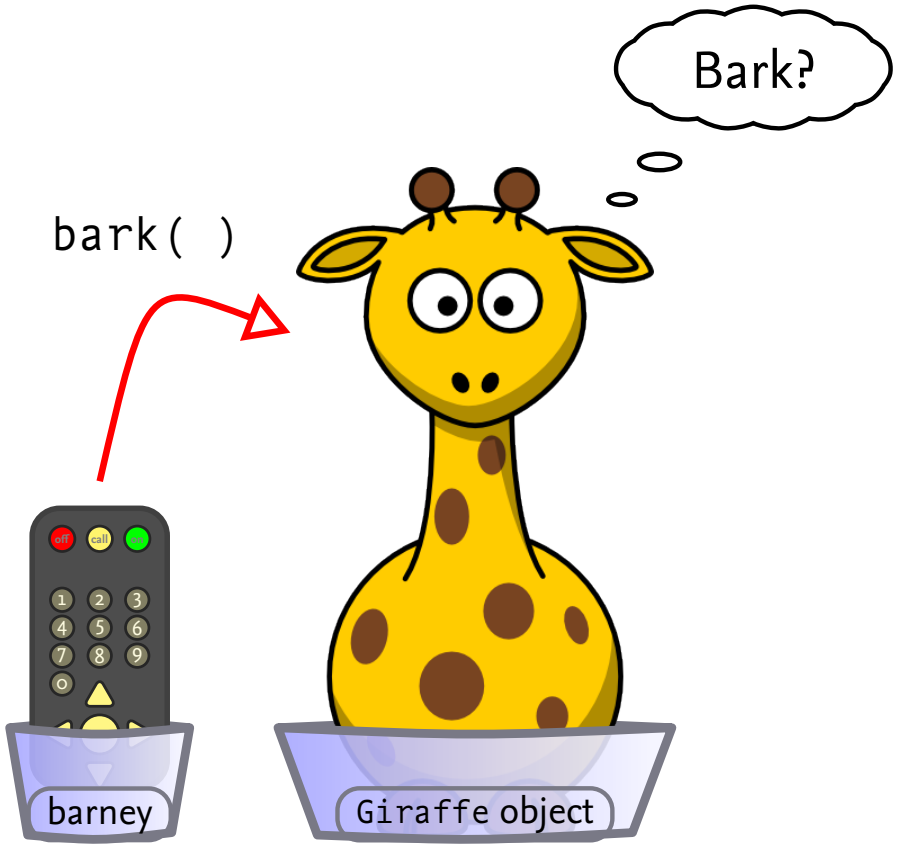
```
Dog barney = new Giraffe( );
barney.bark( );
```



Types Really Matter

Don't Try This at Home

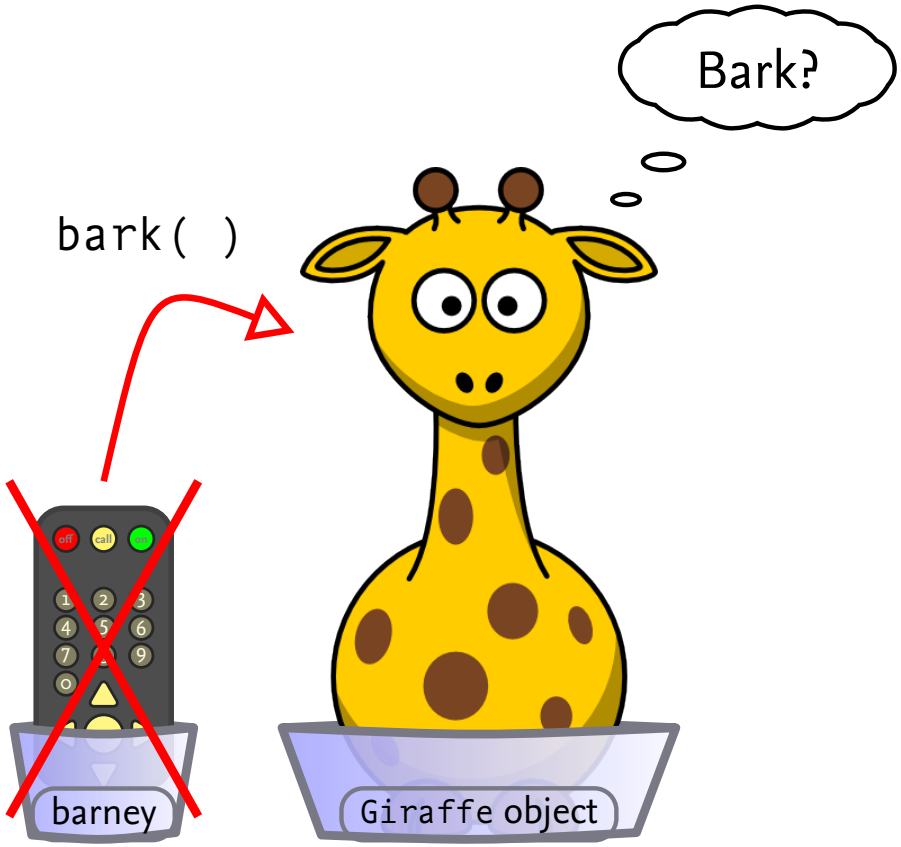
```
Dog barney = new Giraffe( );
barney.bark( ); // ???
```



Types Really Matter

Don't Try This at Home

```
Dog barney = new Giraffe( ); // Impossible
barney.bark( ); // ???
```



Aliases

- Assume you have two `int` variables, `original` and `copy`.
- Assume you do the following:
  - `int original = 3;`
  - `int copy = original;`
- You now expect that `original` and `copy` have the same value: `3`.

Aliases

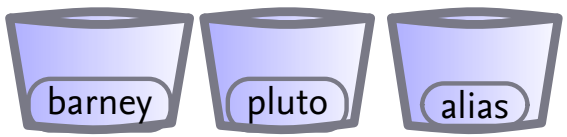
- Assume you have two `int` variables, `original` and `copy`.
- Assume you do the following:
  - `int original = 3;`
  - `int copy = original;`
- You now expect that `original` and `copy` have the same value: `3`.
- For object references this is the same:
  - `BankAccount original = new BankAccount( );`
  - `BankAccount copy = original;`
- After this, `original` and `copy` also have the same value.
  - The value happens to be an object reference value.
  - `original` and `copy` are called *aliases*.
- Now assume we do: `original.deposit( 3.0 );`
  - Then `original.getBalance( )` should return `3.0`.
  - And *so* should `copy.getBalance( )`.
    - Because `original` and `copy` reference the same object.

Aliases

Sharing Object Reference

Java

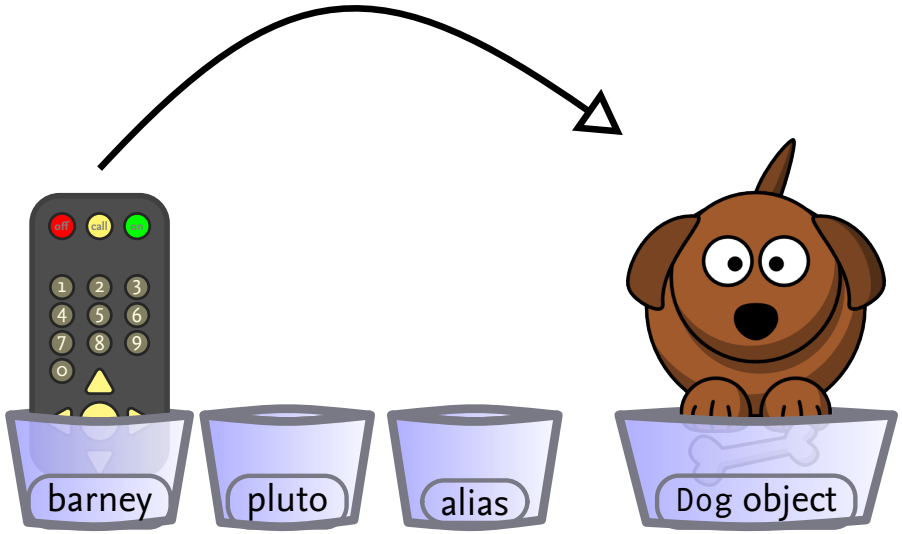
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( );
```



Sharing Object Reference

Java

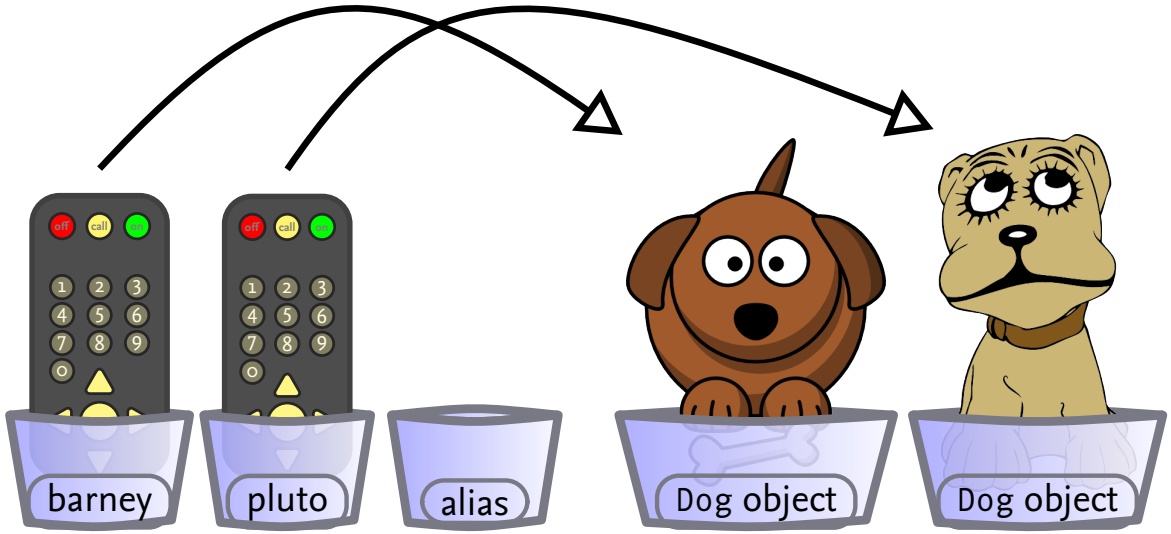
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( );
```



Sharing Object Reference

Java

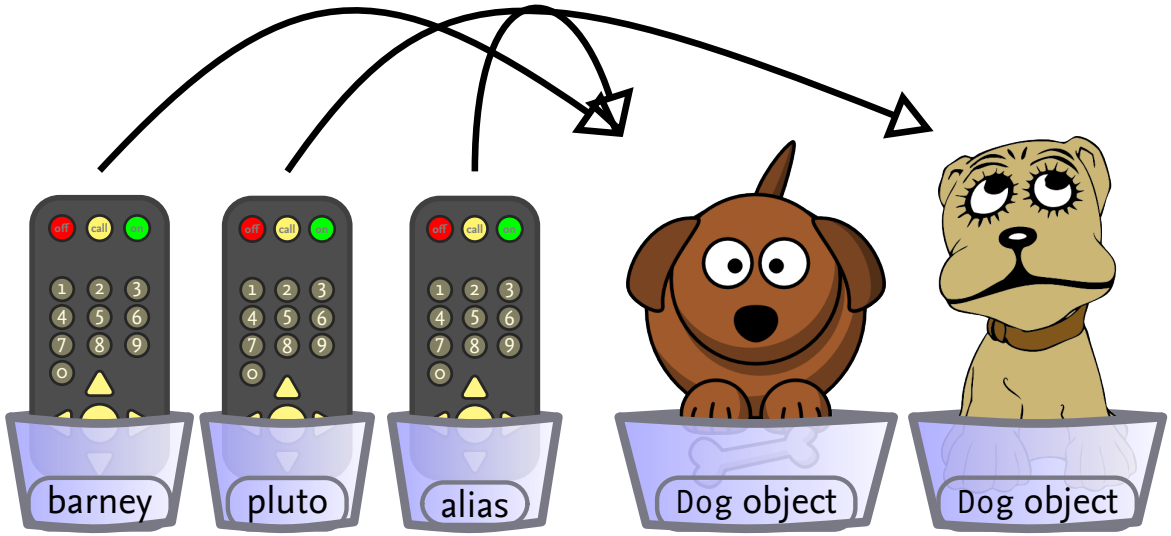
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( );
```



Sharing Object Reference

Java

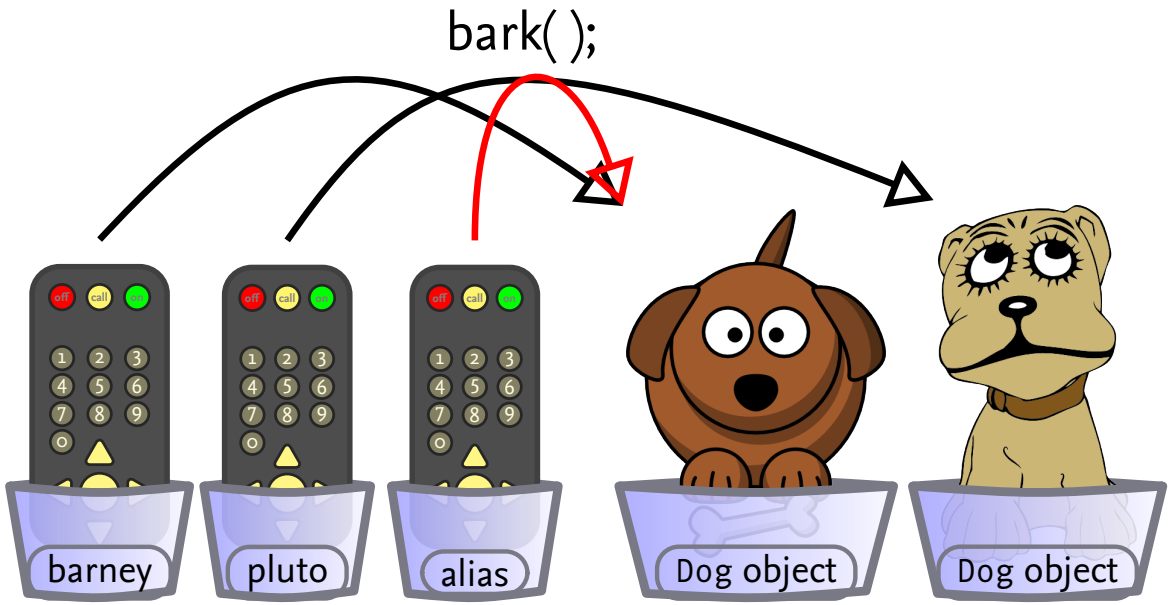
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( );
```



Sharing Object Reference

Java

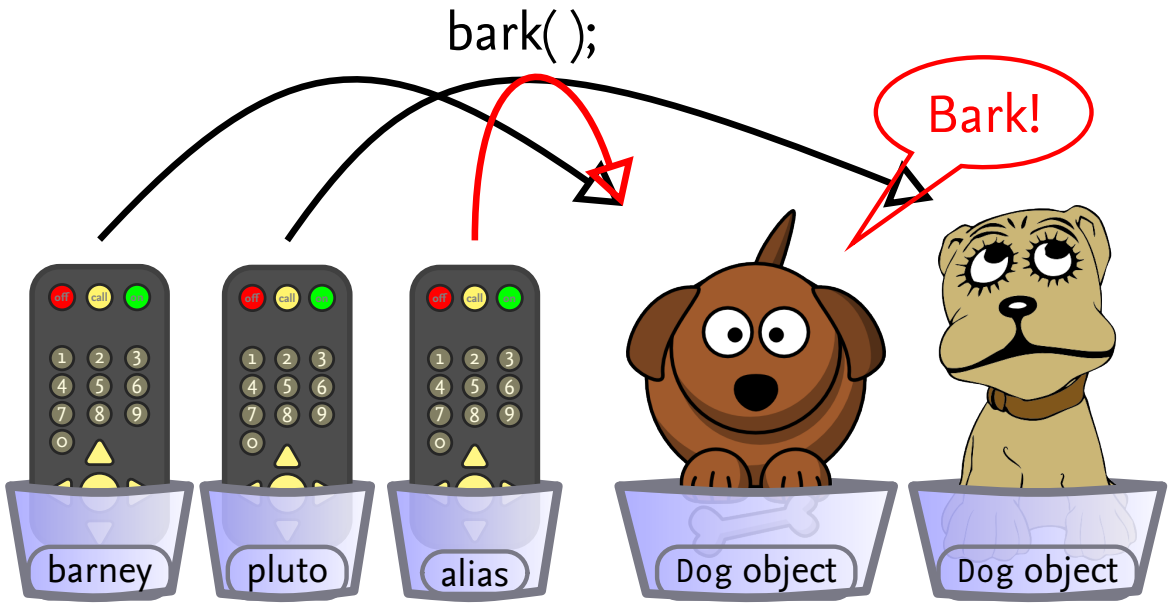
```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( ); // Makes barney's referent bark too.
```



Sharing Object Reference

Java

```
Dog barney = new Dog( );
Dog pluto = new Dog( );
Dog alias = barney;
alias.bark( ); // Makes barney's referent bark too.
```



`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

`null` Values

- Let's assume we have an object reference variable.
  - It may be of any type.
  - `Dog barney;`
- Assume we have a dog object refeference value too.
  - We may assign it to `barney`.
  - `barney = new Dog( );`
- There is one more value you may assign to `barney`.
- The value is an anomaly because it doesn't reference an object.
- The value is has its own keyword: `null`.
  - `barney = null;`
- Remember you need an object reference value if you want to:
  - Access an instance variables; or
  - Call an instance method.
- Only allowed if the object reference value isn't equal to `null`.
- If the reference is equal to `null` you get a runtime exception.

Exception

Don't Try This at Home

```
public class ErrorExample {
    private int attribute;
    public static void main( String[] args ) {
        ErrorExample reference = null;
        reference.attribute = 1;
    }
}
```

Unix Session

```
$
```

Exception

Don't Try This at Home

```
public class ErrorExample {
    private int attribute;
    public static void main( String[] args ) {
        ErrorExample reference = null;
        reference.attribute = 1;
    }
}
```

Unix Session

```
$ javac ErrorExample.java
```

Exception

Don't Try This at Home

```
public class ErrorExample {
    private int attribute;
    public static void main( String[] args ) {
        ErrorExample reference = null;
        reference.attribute = 1;
    }
}
```

Unix Session

```
$ javac ErrorExample.java
$
```

Exception

Don't Try This at Home

```
public class ErrorExample {
    private int attribute;
    public static void main( String[] args ) {
        ErrorExample reference = null;
        reference.attribute = 1;
    }
}
```

Unix Session

```
$ javac ErrorExample.java
$ java ErrorExample
```

Exception

Don't Try This at Home

```
public class ErrorExample {
    private int attribute;
    public static void main( String[] args ) {
        ErrorExample reference = null;
        reference.attribute = 1;
    }
}
```

Unix Session

```
$ javac ErrorExample.java
$ java ErrorExample
Exception in thread "main" java.lang.NullPointerException
        at ErrorExample.main(ErrorExample.java:5)
$
```

The `this` Reference

- In an instance method you can reference the "current" object.
- It's the object whose reference you needed to call the method:
  - `barney.bark( )`
- The instance method needs to know about the reference.
- So how does that work?
- The JVM passes the reference as an implicit extra parameter.
- Inside the method, you can get the reference by writing `this`.
  - (This also works in the constructor; but be careful.)

The this Reference

Java

```
public class ThisExample {
    private int value;

    public ThisExample( final int initialValue ) {
        value = initialValue;
    }

    public static void main( String[] args ) {
        ThisExample reference = new ThisExample( 2 );
        reference.method( );
    }

    public void method( ) {
        System.out.println( "The value is " + value );
    }
}
```

The this Reference

Java

```
public class ThisExample {
    private int value;

    public ThisExample( final int initialValue ) {
        this.value = initialValue;
    }

    public static void main( String[] args ) {
        ThisExample reference = new ThisExample( 2 );
        reference.method( );
    }

    public void method( ) {
        System.out.println( "The value is " + this.value );
    }
}
```

When a Variable Shadows An Attribute

The Variable and Attribute Have Different Names

Java

```
public class ThisExample {
    private int value;

    public ThisExample( final int initialValue ) {
        value = initialValue;
    }

    public static void main( String[] args ) {
        ThisExample reference = new ThisExample( 2 );
        reference.method( );
    }

    public void method( ) {
        System.out.println( The value is " + value );
    }
}
```

When a Variable Shadows An Attribute

The Variable Shadows the Attribute

Java

```
public class ThisExample {
    private int value;

    public ThisExample( final int value ) {
        value = value; // ????
    }

    public static void main( String[] args ) {
        ThisExample reference = new ThisExample( 2 );
        reference.method( );
    }

    public void method( ) {
        System.out.println( The value is " + value );
    }
}
```

When a Variable Shadows An Attribute

Java

```
public class ThisExample {
    private int value;

    public ThisExample( final int value ) {
        this.value = value;
    }

    public static void main( String[] args ) {
        ThisExample reference = new ThisExample( 2 );
        reference.method( );
    }

    public void method( ) {
        System.out.println( The value is " + value );
    }
}
```

For Wednesday

- Study Sections 2.1–2.5
- Read Sections 2.6–2.9.
- Implement unit tests for `getBalance( )` and for `deposit( )`.

Acknowledgements

- This lecture corresponds to [*Big Java, Early Objects*, 2.6–2.9].

Bibliography


Horstfmann, Cay S. *Big Java, Early Objects.* International Student Version. Wiley. ISBN: 978-1-118-31877-5.

About this Document

- This document was created with `pdflatex`.
- The LaTeX document class is `beamer`.