Java:

Learning to Program with Robots

Chapter 08: Collaborative Classes

Chapter Objectives

After studying this chapter, you should be able to:

- Write a class that uses references to an object with instance variables, parameter variables, and temporary variables
- Draw class diagrams depicting collaborating classes
- Explain how reference variables are different from primitive variables
- Explain what an alias is and what dangers arise from aliasing
- Write code that compares two objects for equivalence
- Throw and catch exceptions
- Use a Java collection object to collaborate with many objects, all having the same type

8.1.1: Using a Single Class to Model a Person

Person
-String name -String mother -String father -int birthYr -int birthMth -int birthDay -int deathYr -int deathMth -int deathDay
+Person(String aName, String dad, String mom, int bYear, int bMonth, int bDay) +Person(String aName, String dad, String mom, int bYear, int bMonth, int bDay, int dYear, int dMonth, int dDay) +int daysLived( ) +String getFather( ) +String getMother( ) +String getName( ) +void setDeathDate(int dYear, int dMonth, int dDay) . . .

This is not a good design because:

- Data for a year, month, and day always appear to go together:
  - Triples of parameters
  - Triples of instance variables

  Always grouped with "birth" or "b" or "death" or "d" in the name.
- A separate class would allow these to be grouped into a natural concept, "Date". Doing so:
  - Reduces clutter in the **Person** class
  - Allows dates to be easily used in other programs
  - Could simplify algorithms such as **daysLived**

This class already exists in the **becker** library. A small test program:

```
import becker.util.DateTime;

public class DateTest
{
   public static void main(...)
   {  // October 1, 1990
      DateTime lukesBD =
                  new DateTime(1990, 10, 1);

      // The data and time given by the computer's clock.
      DateTime today = new DateTime();

      int daysOld = lukesBD.daysUntil(today);
      System.out.println("Luke is " + daysOld + " days old.");
   }
}
```

DateTime
-int year -int month -int day
+DateTime( ) +DateTime(int yr, int mth, int day) +DateTime(DateTime dateToCopy) +void addYears(int howMany) +void addMonths(int howMany) +void addDays(int howMany) +int daysUntil(DateTime d) +boolean equals(Object obj) +String format( ) +int getYear( ) +int getMonth( ) +int getDay( ) +boolean isAfter(DateTime d) +boolean isBefore(DateTime d) +void setFormatInclude(int what) +void setFormatLength(int len) +String toString( )

8.1.2: Reimplementing the Person Class (1/2)

```
import becker.util.Test;
import becker.util.DateTime;

public class Person extends Object
{  private String name;                  // person's name
   private String mother;                // person's mother's name
   private String father;                // person's father's name
   private DateTime birth;               // birth date
   private DateTime death;               // death date (null if still alive)

   /** Construct a new person. */
   public Person(String aName, String mom, String dad,
         DateTime birthDate, DateTime deathDate)
   {  super();
      this.name = aName;
      this.mother = mom;
      this.father = dad;

      this.birth = birthDate;
      this.death = deathDate;
   }
```

Instance variables

Parameter variables

Initialize instance variables

8.1.2: Reimplementing the Person Class (2/2)

```
/** Calculate the number of days this person has lived. */
public int daysLived()
{  DateTime endDate = this.death;
   if (this.death == null)
   {  endDate = new DateTime();
   }
   return this.birth.daysUntil(endDate);
}

/** Set the death date to a new value. */
public void setDeathDate(int dYear, int dMo
{  this.death = new DateTime(dYear, dMonth
}

/** Set the death date to a new value. */
public void setDeathDate(DateTime dDate)
{  this.death = dDate;
}

public static void main(String[ ] args)
{  Person p = new Person("Joseph Becker", "Jacob B. Becker",
        "Elizabeth Unruh", new DateTime(1900, 6, 14), null);
   p.setDeathDate(1901, 6, 14);
   Test.ckEquals("exactly 1 year", 365, p.daysLived());
```

Temporary variable

null means “no death date object exists”

Call one of the **Date-Time** services; pass an object reference as an argument

Overload the method

Partial test suite

Assigning the special value **null** to a variable means it does not refer to any object.

A variable can be compared for equality to **null**:

```
if (this.death == null)
{  …
```

```
if (this.death != null)
{  …
```

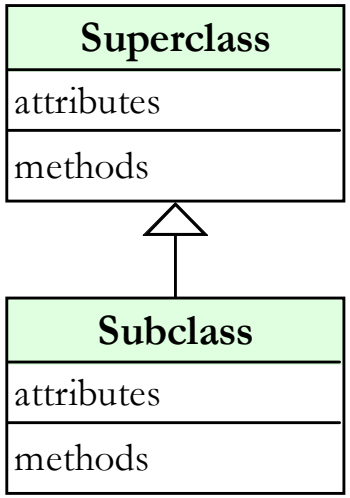
The **<**, **<=**, **=>**, and **>** operators don't work.

Calling a method when a variable contains **null** results in an exception:

```
public class Person extends Object
{  private DateTime death = null;
   …
   public int daysSinceDeath()
   {  DateTime today = new DateTime();
      return this.death.daysUntil(today);
   }
}
```

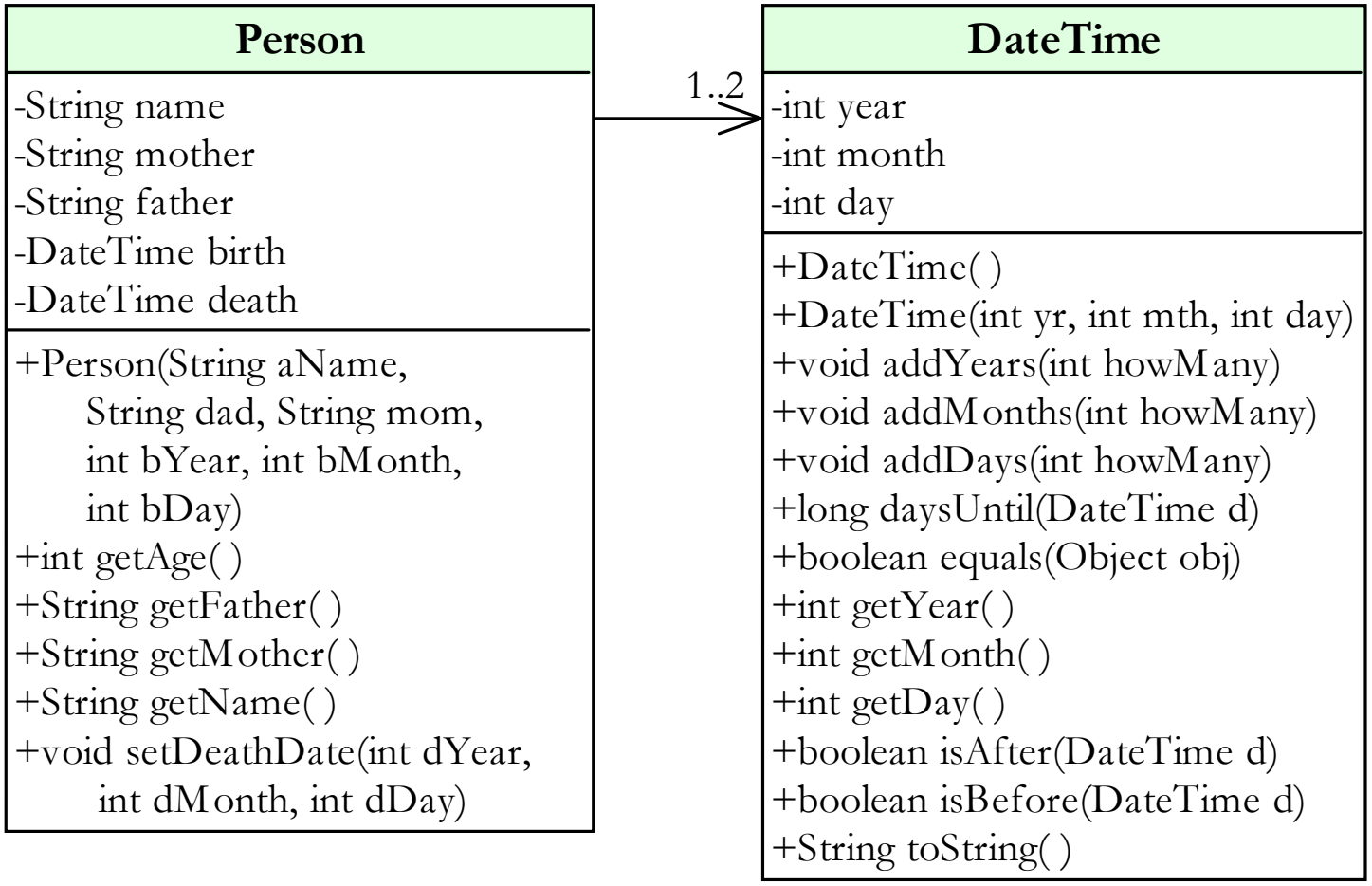
```
Exception in thread "main" java.lang.NullPointerException
        at Person.daysSinceDeath(Person.java:53)
        at Person.main(Person.java:75)
```

8.1.3: Diagramming Collaborating Classes



Two different ways for classes to collaborate;

Two different ways to diagram them



Object references can also be returned by a method.

```
public class Person extends Object
{  …
   private DateTime birth;          // birth date
   private DateTime death;          // death date (null if still alive)
   …
   public DateTime getBirthDate()
   {  return this.birth;
   }
}
```

It can be used in various ways:

```
Person luke = …
Person caleb = …
DateTime lukesBD = luke.getBirthDate();
if (lukesBD.isBefore(caleb.getBirthDate())
{  System.out.println("Luke is older. ");
} else if (caleb.getBirthDate().isBefore(lukesBD))
{  System.out.println("Caleb is older.");
}  else
{  System.out.println("Luke and Caleb are the same age.");
}
```

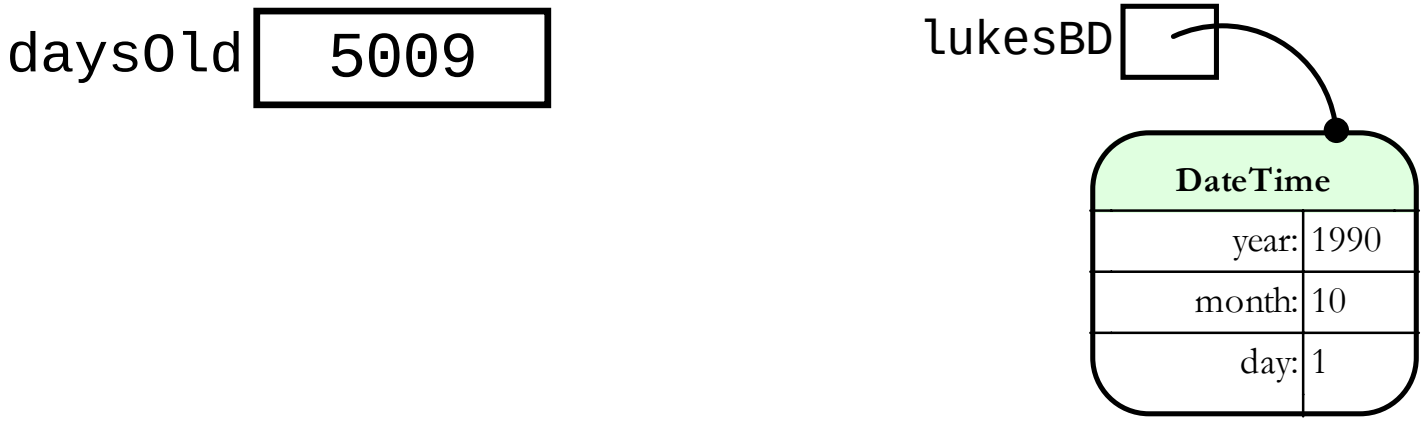
Result assigned to a variable

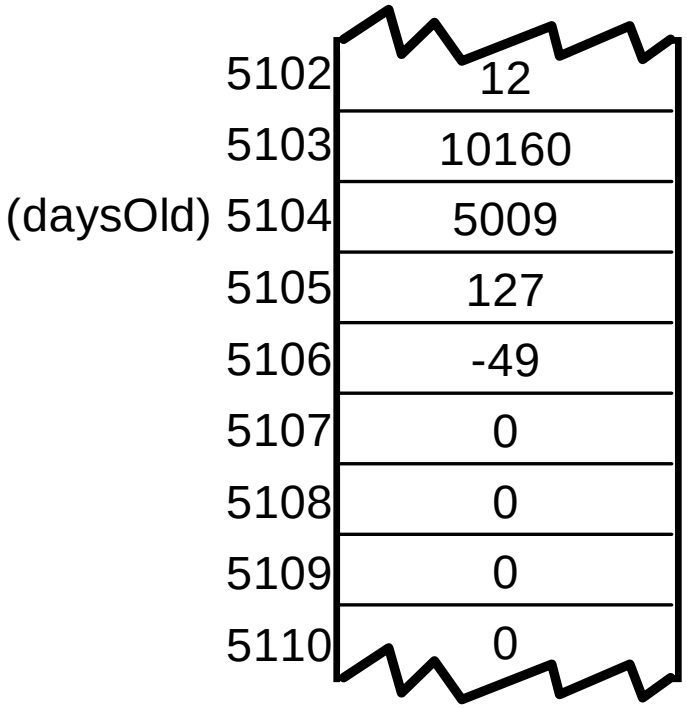
Result passed as an argument

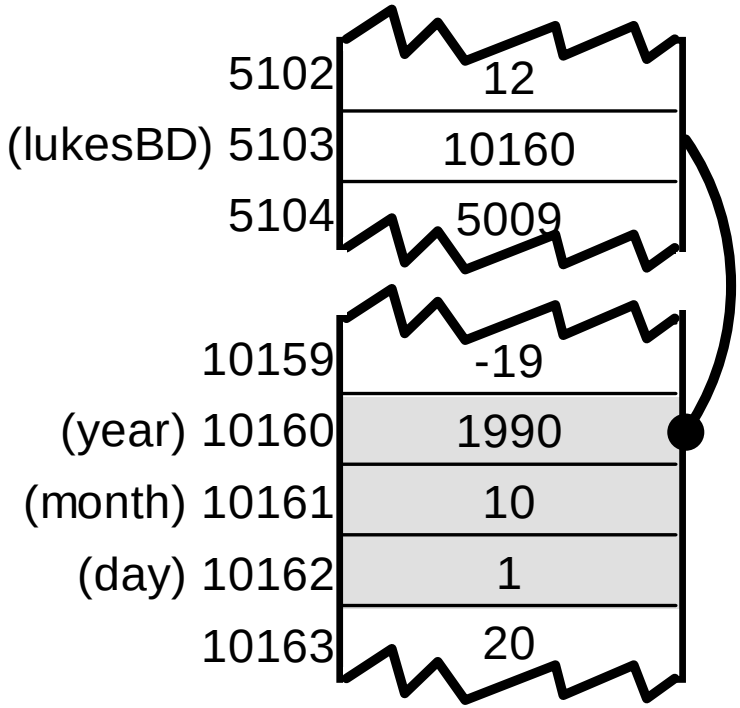
Method calls can be cascaded together

```
import becker.util.DateTime;

public class Main extends Object
{  public static void main(String[ ] args)
   {
      DateTime lukesBD = new DateTime(1990, 10, 1);
      DateTime today = new DateTime();

      int daysOld = lukesBD.daysUntil(today);
      System.out.println("Luke is " + daysOld + " days old.");
   }
}
```

These variables are often represented this way. Why?





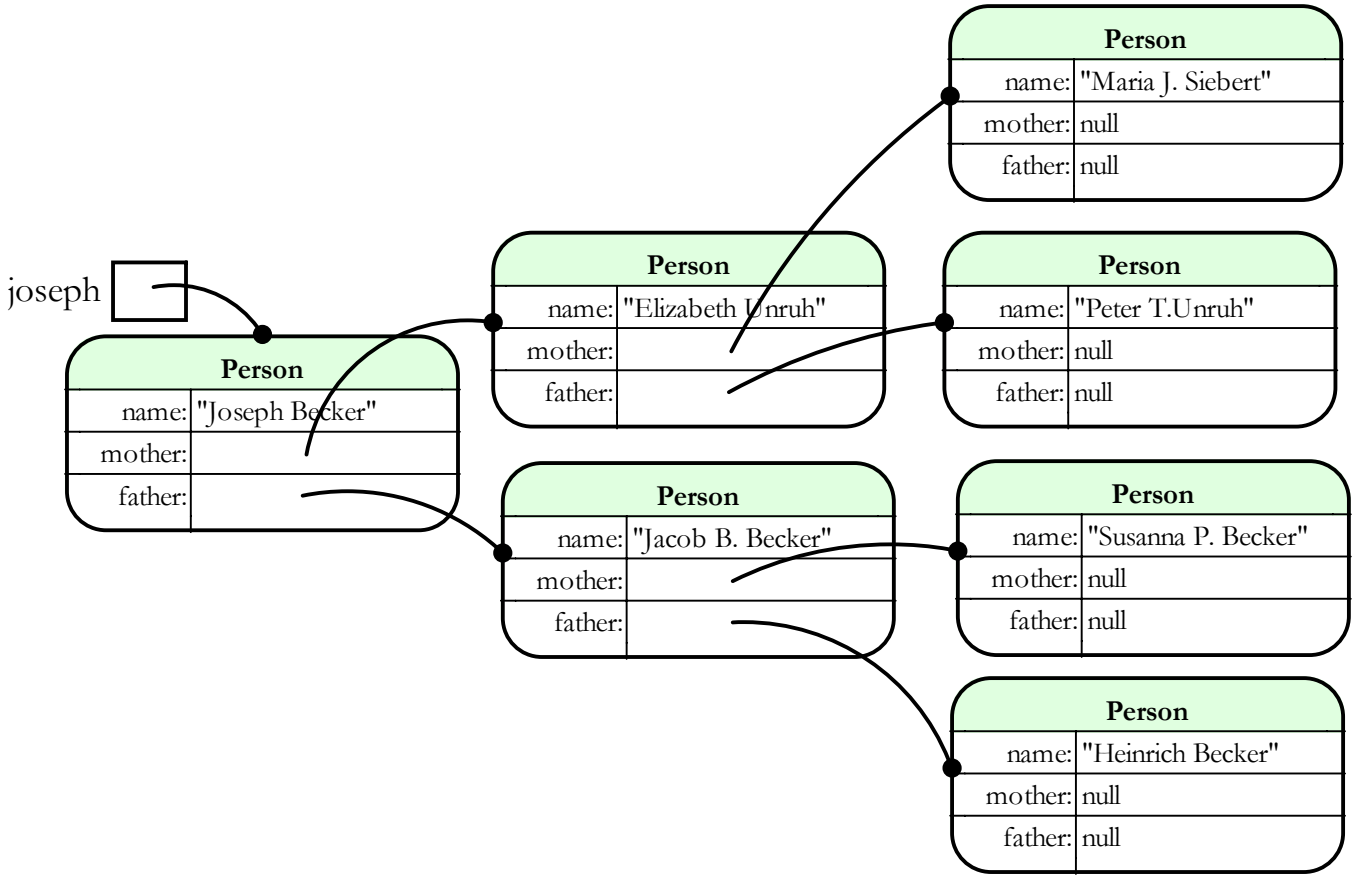


A computer's memory:

- like a long series of boxes, each with an address and holding a value; addresses are numbered consecutively
- a variable (**daysOld**, **lukesBD**) is a name for an address in memory
  - for primitive variables (**int**), it's the address of the value
  - for reference variables (**lukesBD**), it's the address of the address

Implications for reference variables:

- An address is smaller than an object (sometimes *much* smaller) and is easier/faster to assign to a variable and pass to a parameter.
- Objects can be linked together in interesting ways.

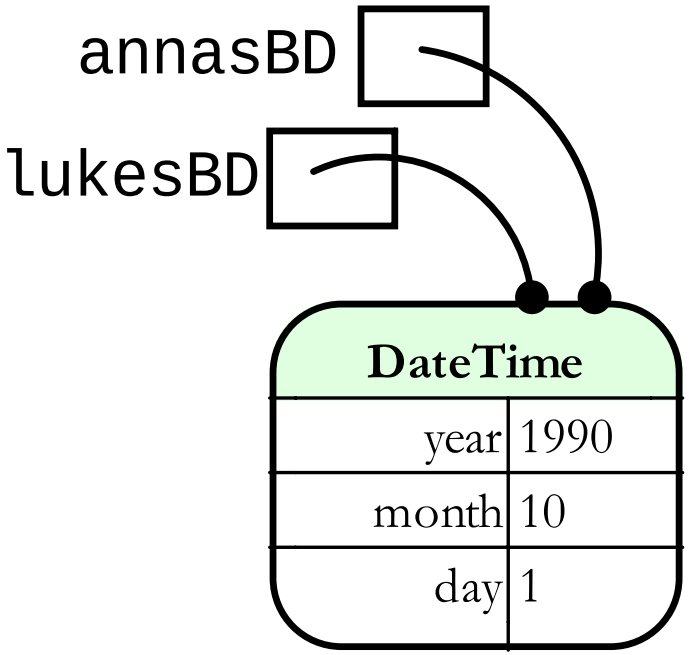


- Several variables can refer to the same object (aliasing).

The following code:

```
DateTime lukesBD = new DateTime(1990, 10, 1);
DateTime annasBD = lukesBD;
```

results in this situation:

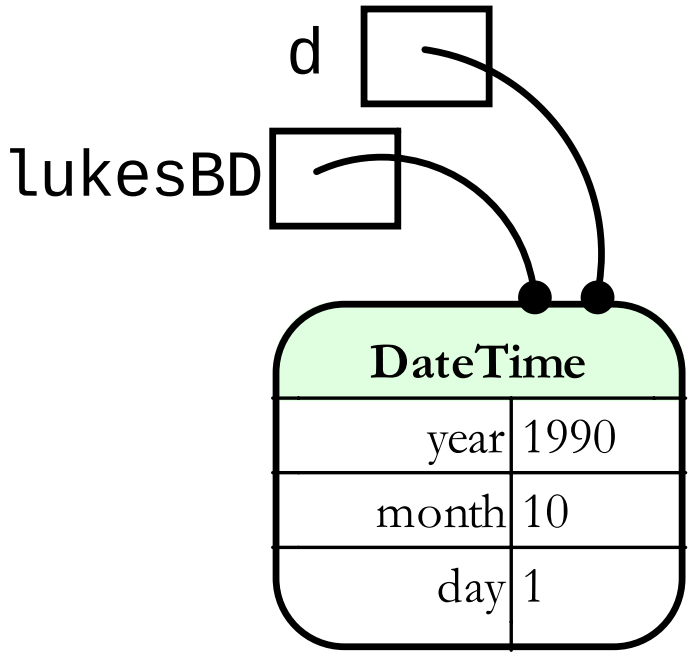



```
lukesBD.addYear(1);
annasBD.addYear(2);
```

results in the **year** instance variable being 1993!

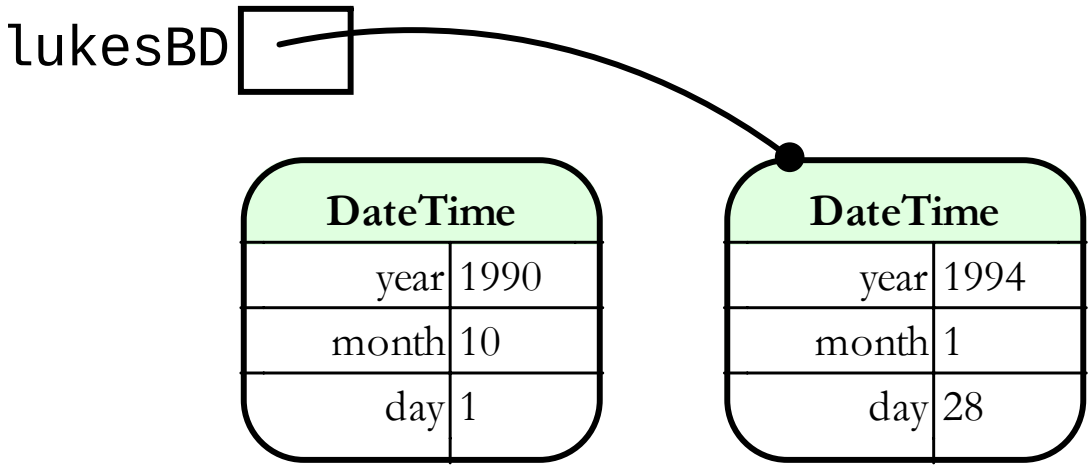
The situation is similar with parameters.

```
DateTime lukesBD = new DateTime(1990, 10, 1);
this.doSomething(lukesBD);
…
public void doSomething(DateTime d)
{ …
```




```
}
```

```
DateTime lukesBD = new DateTime(1990, 10, 1);
...
lukesBD = new DateTime(1994, 1, 28);
```

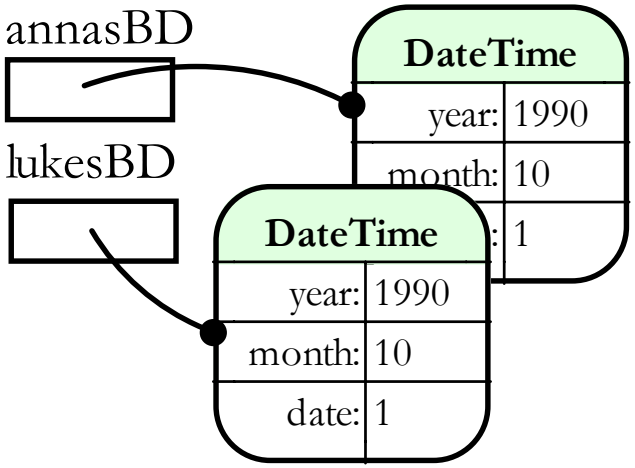


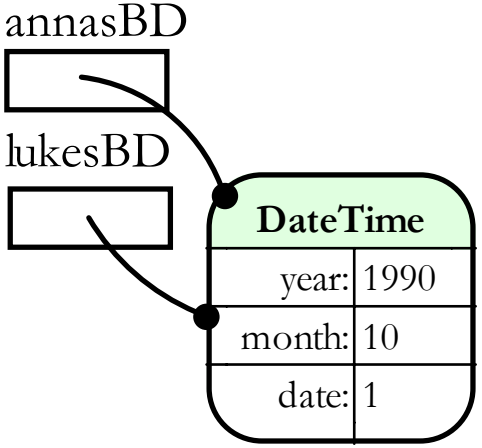
```
DateTime lukesBD = new DateTime(1990, 10, 1);
DateTime annasBD = new DateTime(1990, 10, 1);
DateTime today = new DateTime();

int lukesAge = lukesBD.daysUntil(today);
int annasAge = annasBD.daysUntil(today);

if (lukesAge == annasAge)
{  // what to do if Luke and Anna are the same age
   …


if (lukesBD == annasBD)
{  // what to do if Luke and Ann have the same birth date
   …
```

Returns **true**

Returns **false**

Why?





8.2.4 A Method to Test Equivalence

```
public class DateTime extends Object
{  private int year;
   private int month;
   private int day;

   ...
   /** Test if this date/time object means the same date/time as the other DateTime object. */
   public boolean isEquivalent(DateTime other)
   {  return other != null &&
              this.year == other.getYear() &&
              this.month == other.getMonth() &&
              this.day == other.getDay();
   }

   /** Another way to do it. */
   public boolean isEquivalent(DateTime other)
   {  return other != null &&
              this.year == other.year &&
              this.month == other.month &&
              this.day == other.day;
   }
}
```

Case Study: Vote Counter

Write an electronic vote counter to use in an election. The user interface is provided, as shown. One window is for the voter to make his or her selection. The other window is used by the official at the polling station to open and close the election and to enable the next voter to vote.

You will need to provide a class that implements the **IVoteCounter** interface (see next slide).

For now, your class must support between 1 and 4 candidates. We will remove that restriction soon.

Case Study: IVoteCounter Interface (1/2)

```
import becker.util.IView;

/** Objects implementing this interface can count the votes in an election.
*
*  @author Byron Weber Becker */
public interface IVoteCounter
{
    /** Add a candidate for the election.
    *  @param name The candidate's name
    *  @param pixFile A gif or jpg of the candidate, if available */
    public void addCandidate(String name, String pixFile);

    /** Get the number of candidates registered with this vote counter. */
    public int getNumCandidates();

    /** Get the name of the give candidate.
    *  @param candidateNum The number (0..getNumCandidates()-1) of the candidate whose
    *                      name is desired. */
    public String getName(int candidateNum);

    /** Get the picture file name of the give candidate.
    *  @param candidateNum The number (0..getNumCandidates()-1) of the candidate whose
    *                      pixfile is desired. */
    public String getPixFile(int candidateNum);
```

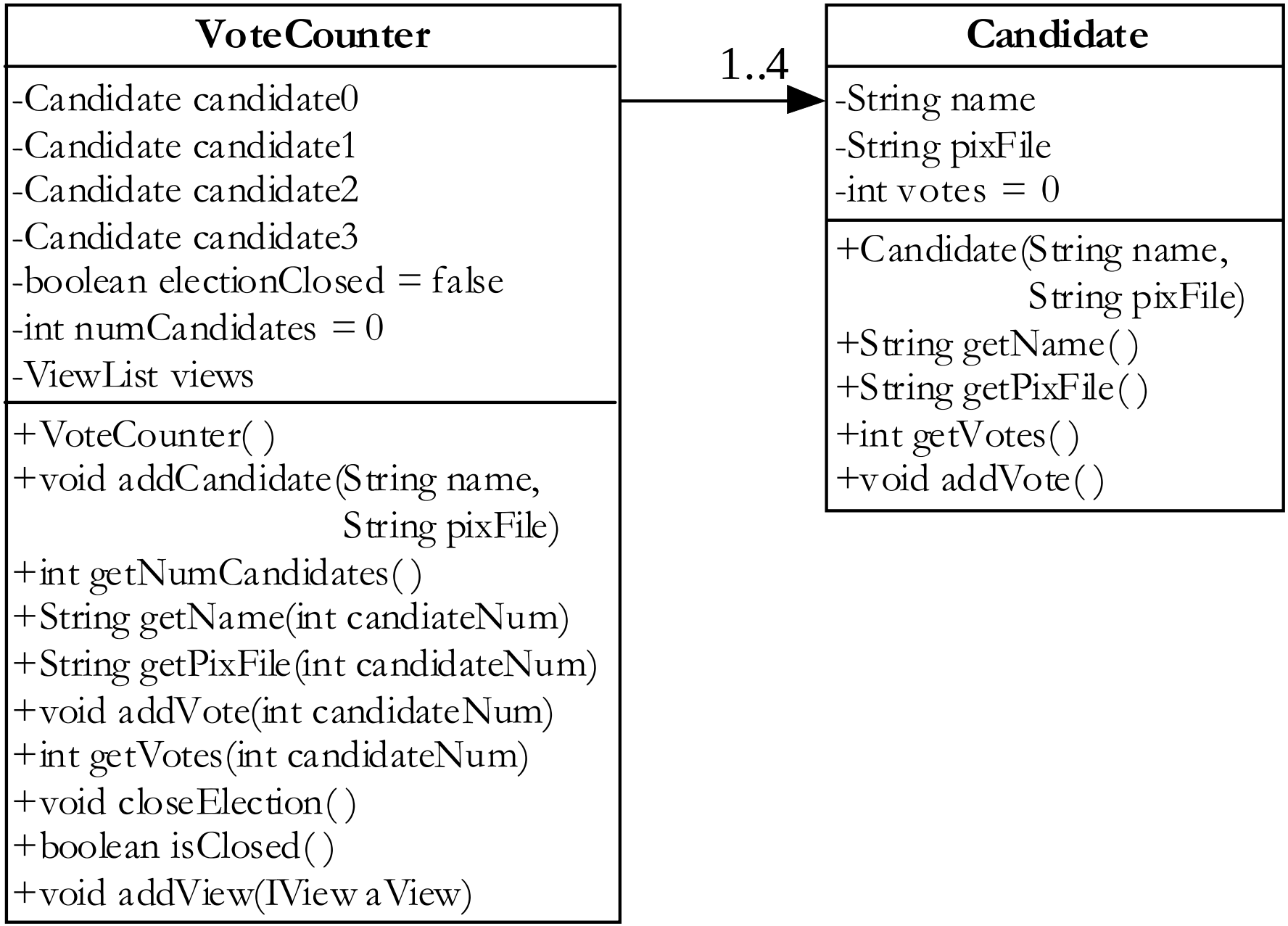
Case Study: IVoteCounter Interface (2/2)

```
    /** Add one vote for the named candidate, provided the vote has not yet closed..
    *  @param candidateNum The number (0..getNumCandidates()-1) of the candidate that
    *              received the vote. */
    public void addVote(int candidateNum);

    /** Get the number of votes for the named candidate, but only if the election has closed.
    * @param candidateName The name of a candidate
    * @return The number of votes received by the candidate; -1 if voting has not yet closed. */
    public int getVotes(int candidateNum);

    /** Stop accepting votes for this election. */
    public void closeElection();

    /** Is the election closed? */
    public boolean isClosed();

    /** Add a view to this model.
     *  @param aView The view to add. */
    public void addView(IView aView);
}
```

Case Study: A Poor Class Design

VoteCounter
-String candName0 -String candName1 -String candName2 -String candName3 -String picture0 -String picture1 -String picture2 -String picture3 -int votes0 = 0 -int votes1 = 0 -int votes2 = 0 -int votes3 = 0 -boolean electionClosed -int numCandidates = 0 -ViewList views
+VoteCounter( ) +void addCandidate(String name, String pixFile) +int getNumCandidates( ) +String getName(int candidateNum) +String getPixFile(int candidateNum) +void addVote(int candidateNum) +int getVotes(int candidateNum) +void closeElection( ) +boolean isClosed( ) +void addView(IView aView)

Case Study: A Much Better Design



Case Study: Implementing Candidate (1/2)

```
import becker.util.Test;

public class Candidate extends Object
{
   public Candidate(String name, String pixFile)…
   public String getName()       {  return "dummy name";}
   public String getPixFile()    {  return "dummy pixFile";  }
   public int getVotes()         {  return 0;  }
   public void addVote()         {  }

   public static void main(String[ ] args)
   {  Candidate c = new Candidate("John Doe", "doe.jpg");

      Test.ckEquals("name", "John Doe", c.getName());
      Test.ckEquals("pixFile", "doe.jpg", c.getPixFile());
      Test.ckEquals("no votes yet", 0, c.getVotes());

      c.addVote();
      c.addVote();
      Test.ckEquals("2 votes", 2, c.getVotes());
   }
}
```

Case Study: Implementing Candidate (2/2)

```
import becker.util.Test;

public class Candidate extends Object
{  private String name;
   private String pixFile;
   private int numVotes = 0;

   public Candidate(String name, String pixFile)
   {  super();
      this.name = name;
      this.pixFile = pixFile;
   }

   public String getName()       {  return this.name;       }

   public String getPixFile()    {  return this.pixFile;    }

   public int getVotes()         {  return this.numVotes;  }

   public void addVote()         {  this.numVotes += 1;     }

   public static void main(String[ ] args)
   {  …
   }
}
```

Case Study: Testing VoteCounter (1/2)

```
import becker.util.Test;
import becker.util.ViewList;
import becker.util.IView;

public class VoteCounter extends Object implements IVoteCounter
{
   // Implement stubs here…

   public static void main(String[ ] args)
   {  VoteCounter vc = new VoteCounter();

      Test.ckEquals("new", 0, vc.getNumCandidates());
      Test.ckEquals("not closed", false, vc.isClosed);

      vc.addCandidate("Karen Redman", "redman.jpg");
      Test.ckEquals("added redman", 1, vc.getNumCandidates());

      vc.addCandidate("Stephen Harper", "harper.jpg");
      Test.ckEquals("added harper", 2, vc.getNumCandidates());

      vc.addCandidate("Alexa McDonough", "mcdonough.jpg");
      Test.ckEquals("added mcdonough", 3, vc.getNumCandidates());

      vc.addCandidate("Gilles Duceppe", "duceppe.jpg");
      Test.ckEquals("added duceppe", 4, vc.getNumCandidates());
```

Case Study: Testing VoteCounter (2/7)

```
        Test.ckEquals("check name", "Stephen Harper", vc.getName(1));
        Test.ckEquals("check name", "Alexa McDonough", vc.getName(2));
        Test.ckEquals("check pixFile", "mcdonough.jpg", vc.getPixFile(2));
        Test.ckEquals("check votes", -1, vc.getVotes(2)); // not closed yet

        vc.addVote(1);
        vc.addVote(2);
        vc.addVote(2);
        vc.addVote(3);
        vc.addVote(3);
        vc.addVote(3);
        vc.closeElection();
        Test.ckEquals("check votes", 0, vc.getVotes(0));
        Test.ckEquals("check votes", 1, vc.getVotes(1));
        Test.ckEquals("check votes", 2, vc.getVotes(2));
        Test.ckEquals("check votes", 3, vc.getVotes(3));

        // election closed; shouldn't be recorded
        vc.addVote(0);
        Test.ckEquals("check votes", 0, vc.getVotes(0));
    }
}
```

Case Study: Implementing VoteCounter (1/5)

```
import becker.util.Test;
import becker.util.ViewList;
import becker.util.IView;

/** Count the votes for up to four candidates in an election.
*
* @author Byron Weber Becker */
public class VoteCounter extends Object implements IVoteCounter
{
   private Candidate cand0;
   private Candidate cand1;
   private Candidate cand2;
   private Candidate cand3;
   private int numCandidates = 0;
   private boolean isClosed = false;

   private ViewList views = new ViewList();

   /** Construct a new vote counter object. */
   public VoteCounter()
   {  super();
   }
```

Case Study: Implementing VoteCounter (2/5)

```
/** Add a candidate for the election.
*  @param name The candidate's name
*  @param pixFile A gif or jpg of the candidate, if available */
public void addCandidate(String name, String pixFile)
{  if (this.numCandidates == 0)
   {  this.cand0 = new Candidate(name, pixFile);
      this.numCandidates += 1;
   } else if (this.numCandidates == 1)
   {  this.cand1 = new Candidate(name, pixFile);
      this.numCandidates += 1;
   } else if (this.numCandidates == 2)
   {  this.cand2 = new Candidate(name, pixFile);
      this.numCandidates += 1;
   } else if (this.numCandidates == 3)
   {  this.cand3 = new Candidate(name, pixFile);
      this.numCandidates += 1;
   } else
   {  System.out.println("Error: too many candidates");
   }

   this.views.updateAllViews();
}
```

Case Study: Implementing VoteCounter (3/5)

```
/** Get the number of candidates registered with this vote counter. */
public int getNumCandidates()       {  return this.numCandidates;  }

/** Get the name of the give candidate.
*  @param candidateNum The number of the candidate whose name is desired. */
public String getName(int candidateNum)
{  Candidate c =  this.getCandidate(candidateNum);
   return c.getName();
}

/** Get the picture file name of the give candidate.
*  @param candidateNum The number of the candidate whose pixfile is desired. */
public String getPixFile(int candidateNum)
{  return this.getCandidate(candidateNum).getPixFile();
}

/** Add one vote for the named candidate, provided the vote has not yet closed..
*  @param candidateNum The number of the candidate that received the vote. */
public void addVote(int candidateNum)
{  if (!this.isClosed)
   {  this.getCandidate(candidateNum).addVote();
      this.views.updateAllViews();
   }
}
```

Case Study: Implementing VoteCounter (4/5)

```
/** Get the number of votes for the named candidate, but only if the voting has closed.
* @param candidateName The name of a candidate
* @return The number of votes received by the candidate; -1 if voting has not yet closed. */
public int getVotes(int candidateNum)
{  if (this.isClosed)
   {  return this.getCandidate(candidateNum).getVotes();
   } else
   {  return -1;
   }
}
/** Get the candidate object with the given candidate number. */
private Candidate getCandidate(int candidateNum)
{  Candidate c = null;
   if (candidateNum == 0)
   {  c = this.cand0;
   } else if (candidateNum == 1)
   {  c = this.cand1;
   } else if (candidateNum == 2)
   {  c = this.cand2;
   } else if (candidateNum == 3)
   {  c = this.cand3;
   }
   return c;
}
```

Case Study: Implementing VoteCounter (5/5)

```
    /** Stop accepting votes for this election. */
    public void closeElection()
    {  this.isClosed = true;
       this.views.updateAllViews();
    }

    /** Is the voting closed? */
    public boolean isClosed()
    {  return this.isClosed;
    }

    /** Add a view to this model.
    *  @param aView The view to add. */
    public void addView(IView aView)
    {  this.views.addView(aView);
    }

    public static void main(String[ ] args)
    {  VoteCounter vc = new VoteCounter();

       // Testing, as before
       …
    }
}
```