CSE1030 – Introduction to Computer Science II

Lecture #7

Aggregation & Composition I

Goals for Today

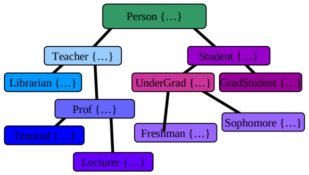
- Goals
  - Theory:
    - “is-a” versus “has-a”
- Practical: (Assignment #3!)
  - “has-a” relationships
  - Collections

CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

Review: Two Things We’ve Already Seen

- Class Hierarchy



- The Person Class

```
public class Person
{
   // attributes
   private String Name;
   private int    Age;
   private int    Weight;

   Person(String name, int age, int weight)
      { Name = name; Age = age; Weight = weight; }

   // methods
   public String getName()         { return Name; }
   public void   setName(String n) { Name = n;    }

   public int  getAge()      { return Age; }
   public void setAge(int a) { Age = a;    }

   public void setWeight(int w) { Weight = w; }
}
```

CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

Big Theory Idea for Today

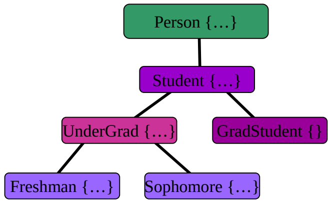
- We have talked about the idea that we should design our classes so that they reflect the **Inherent Relationships** of the problem domain
- Examples:
  - People: Names / Ages
  - Credit Cards: Card #s / Credits / Balances
  - Students: People / UnderGrad / Courses
- What about the relationships between Objects?

Big Theory Idea for Today

- Time to move towards a more formal definition of “Inherent Relationships”
- There are 2 kinds of **Relationships** that occur between objects:
  - “has-a” Relationships
    - People: Names / Ages / Weights
    - Credit Cards: Card #s / Credits / Balances
  - “is-a”
    - Student “is-a” Person
    - visa “is-a” credit card

We have seen both kinds of relationship before…

- “is-a”
  - e.g., Class Hierarchy:



- “has-a”
  - e.g., Person Class:

```
public class Person
{
   // attributes
   private String Name;
   private int    Age;
   private int    Weight;

   Person(String name, int age,
                  int weight)
   {
      Name = name;
      Age = age;
      Weight = weight;
   }
...
```

Implications

- "**is-a**" relationships define the **Class Hierarchy**
  - We haven't talked much about this yet
  - It's coming up soon (next module…)
- "**has-a**" relationships define the **Data Members** (static and instance) that should be contained within a Class or Object
  - We've been using these for a couple of weeks now, although we haven't been using the term "has-a"
  - We have a few more things to say about this…

CSE1030 – Lecture #7

- Review
- Theory: "is-a" versus "has-a"
- Special Case 1: Has 1
- Special Case 2: Has a "Known" Number
- General Case: Collections
- Retrieving Data from a Collection
- We're Done!

When having a single "has-a"

- A person only has a single name, or single age
- Consequently we reflect these "has-a" relationships by including a single data member
- Usually best to keep the functionality simple, as we have in the examples:
  - Private Data
  - Appropriate Accessor / Mutator functions
  - Keep Names Meaningful

Recall The Person Class:

```
public class Person
{
   // attributes
   private String name;
   private int    age;

   // constructor
   Person(String name, int age)
      { this.name = name; this.age = age; }

   // methods
   public String getName() { return name; }
   public void   setName(String name)
      { this.name = name; }

   public int  getAge() { return age; }
   public void setAge(int age)
      { this.age = age; }
}
```

Reminder for William:
Style Suggestions:
javaNamingConvention
CapitalClasses
Don't Forget Comments!

CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

A Small Known Number of “has-a”

- Have to figure-out two things:
  - How to store the data in the class (private data organisation)
    - Arrays (don’t know about yet)
    - Collections (learning about in a few slides)
  - How to design the API to be friendly
- It’s OK to generalise what we already know…

Baseball Fielders

- In Baseball, when a team plays the field, they have exactly 9 players
- This is a “has-a” relationship
  (teams **are not** players, they **have** players)
- What would the corresponding Java Class look like?

```
public class BaseballFielders
{
    private Person pitcher;
    private Person catcher;
    private Person firstBaseman;
    private Person secondBaseman;
    private Person thirdBaseman;
    private Person shortstop;
    private Person LeftFielder;
    private Person centreFielder;
    private Person rightFielder;
```

Reminder for William:
Layout of
Professional Code

```
// constructor
public BaseballFielders(
   Person pitcher,
   Person catcher,
   Person firstBaseman,
   Person secondBaseman,
   Person thirdBaseman,
   Person shortstop,
   Person LeftFielder,
   Person centreFielder,
   Person rightFielder
){
   this.pitcher       = pitcher;
   this.catcher       = catcher;
   this.firstBaseman  = firstBaseman;
   this.secondBaseman = secondBaseman;
   this.thirdBaseman  = thirdBaseman;
   this.shortstop     = shortstop;
   this.LeftFielder   = LeftFielder;
   this.centreFielder = centreFielder;
   this.rightFielder  = rightFielder;
}
```

```
  // accessors
    public Person getPitcher()
       { return new Person(pitcher); }

    // ...



    // mutators
    public void setPitcher(Person pitcher)
       { this.pitcher = new Person(pitcher); }

    // ...
 }
```

The Accesors and Mutators for the rest of the members of the team are omitted for brevity.

CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

What if you don’t know how many?

- Java provides **Collections** to conveniently store an unknown number of objects
- Can store collections of any type of object
- There are 3 main families (types) of collection:
  - Sets
  - Lists
  - Maps

Sets

- Are like the mathematical notion of “set”, or like a shopping list:
  - {Eggs, Milk, Bread, Chocolate, …}
- No Duplicates
- No notion of numerical or alphabetic “order”

Reminder for William:
import

```
import java.util.*;

public class set
{
   public static void main(String[] args)
   {
      // create a set to store my friends
      HashSet<Person> friends = new HashSet<Person>();

      // create some friends
      Person sally = new Person("Sally", 32);
      Person frank = new Person("Frank", 44);
      Person billy = new Person("Billy", 36);

      // add them to my collection
      friends.add(sally);
      friends.add(frank);
      friends.add(billy);

      System.out.println("I have " + friends.size()
                                             + " friends");
   }
}
```

Output

```
> java set
I have 3 friends
```

- For more information about HashSet<>
  http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html
- How to get our objects back out?
  We'll cover that in a slide or two…

Lists

- Are like a "To Do" list, a sequence of objects:
  1. Weekly Readings
  2. Go to Class
  3. Work on Assignment
  4. Send e-mail to Prof telling him how riveting his lectures are
  5. Send e-mail to Prof telling him how riveting his lectures are
  6. Submit Assignment
- Can have Duplicates
- Does have a notion of "order"
  (not necessarily numeric or alphabetic)

```
import java.util.*;

public class list
{
   public static void main(String[] args)
   {
      // list of people I need to visit
      LinkedList<Person> visits = new LinkedList<Person>();

      // create some people to visit
      Person sally = new Person("Sally", 32);
      Person frank = new Person("Frank", 44);
      Person billy = new Person("Billy", 36);

      // construct list of upcoming visits
      visits.add(sally);
      visits.add(frank);
      visits.add(billy);
      visits.add(frank);

      System.out.println("I have planned " + visits.size()
                                             + " visits");
   }
}
```

Duplicates Allowed!

Output

```
> java list
I have planned 4 visits
```

- For more information about LinkedList<>
  http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

Maps

- Are like a dictionary:
  mapping one object (the **key**)
  to another (the **value**)
  - (“Hello” → “Bonjour”)
  - (“My Name Is” → “Je m’appelle”)
  - (“Croissant” → “Croissant”)
- Keys must be Unique,
  Values can be Duplicates

(Key, Value) Pairs

```
import java.util.*;

public class map
{
   public static void main(String[] args)
   {
      // my list of contacts
      HashMap<String,Person> contacts
                        = new HashMap<String,Person>();

      // create some people to visit
      Person sally = new Person("Sally Yeh", 32);
      Person frank = new Person("Frank Sinatra", 44);
      Person billy = new Person("Billy Holiday", 36);

      // construct list of upcoming contacts
      contacts.put("Sally", sally);
      contacts.put("Frank", frank);
      contacts.put("Billy", billy);

      System.out.println("I have " + contacts.size()
                                          + " contacts");
   }
}
```

Output

```
> java map
I have 3 contacts
```

- For more information about HashMap<K,V>
  http://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html

Final Note Regarding Collections

- There are many variations of these Collections:
- Set
  - AbstractSet, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, HashSet, JobStateReasons, LinkedHashSet, TreeSet
- List
  - AbstractList, AbstractSequentialList, ArrayList, AttributeList, CopyOnWriteArrayList, LinkedList, RoleList, RoleUnresolvedList, Stack, Vector
- Map
  - AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, TreeMap, UIDefaults, WeakHashMap

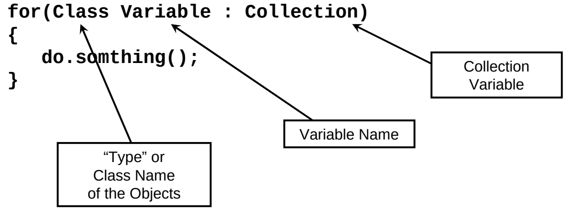
CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

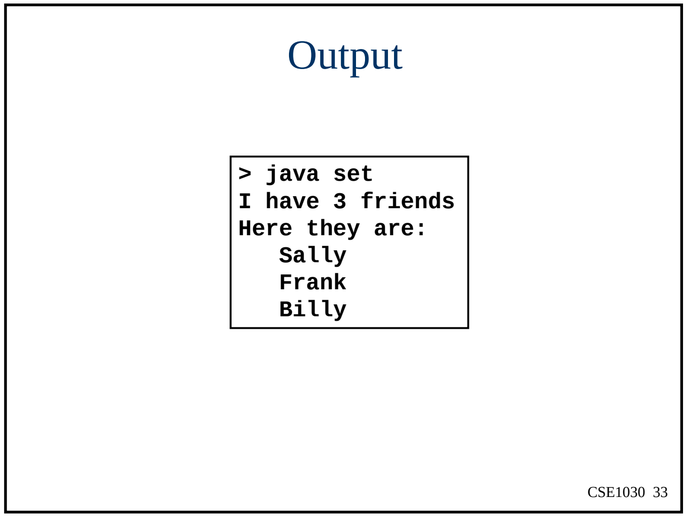
Automatic Iteration

- Automatic Iteration is an easy way to get access to the data stored in a Collection
- In Java code it looks like this:
  - `for(Class Variable : Collection)`

```
{
   do.somthing();
}
```

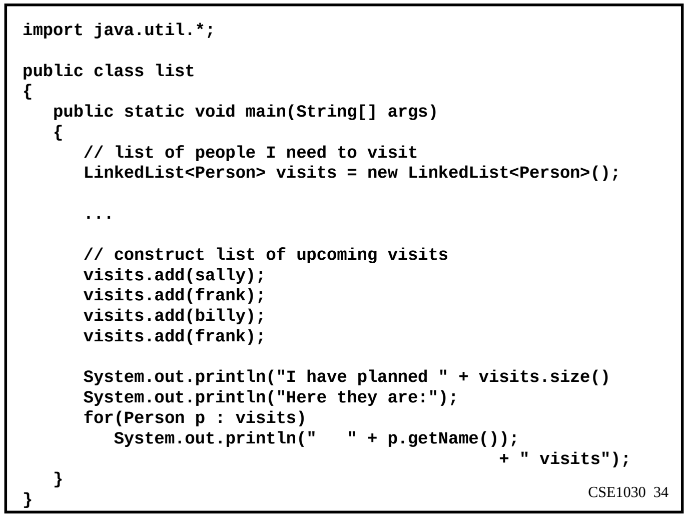


```
import java.util.*;

public class set
{
   public static void main(String[] args)
   {
      // create a set to store my friends
      HashSet<Person> friends = new HashSet<Person>();

      ...

      // add them to my collection
      friends.add(sally);
      friends.add(frank);
      friends.add(billy);

      System.out.println("I have " + friends.size()
                                             + " friends");
      System.out.println("Here they are:");
      for(Person p : friends)
         System.out.println("   " + p.getName());
   }
}
```

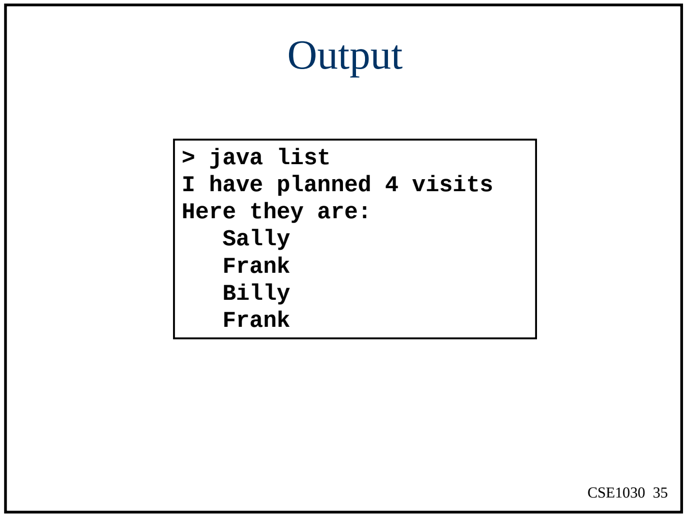


Output

```
> java set
I have 3 friends
Here they are:
   Sally
   Frank
   Billy
```

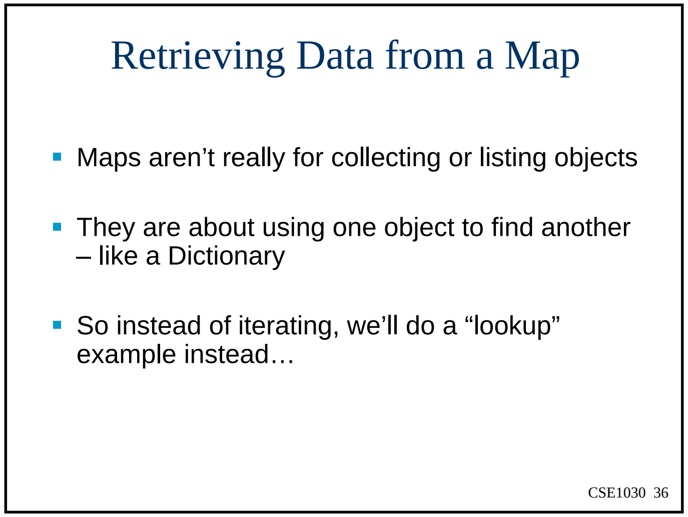


```
import java.util.*;

public class list
{
   public static void main(String[] args)
   {
      // list of people I need to visit
      LinkedList<Person> visits = new LinkedList<Person>();

      ...

      // construct list of upcoming visits
      visits.add(sally);
      visits.add(frank);
      visits.add(billy);
      visits.add(frank);

      System.out.println("I have planned " + visits.size()
      System.out.println("Here they are:");
      for(Person p : visits)
         System.out.println("   " + p.getName());
                                               + " visits");
   }
}
```



Output

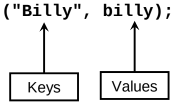
```
> java list
I have planned 4 visits
Here they are:
   Sally
   Frank
   Billy
   Frank
```



Retrieving Data from a Map

- Maps aren’t really for collecting or listing objects
- They are about using one object to find another – like a Dictionary
- So instead of iterating, we’ll do a “lookup” example instead…

```
import java.util.*;

public class map
{
   public static void main(String[] args)
   {
      // my list of contacts
      HashMap<String,Person> contacts
                        = new HashMap<String,Person>();

      // create some people to visit
      Person sally = new Person("Sally Yeh", 32);
      Person frank = new Person("Frank Sinatra", 44);
      Person billy = new Person("Billy Holiday", 36);

      // construct list of upcoming contacts
      contacts.put("Sally", sally);
      contacts.put("Frank", frank);
      contacts.put("Billy", billy);
```



```
      System.out.println("I have " + contacts.size()
                                          + " contacts");

      Person p = contacts.get("Frank");
      if(p != null)
         System.out.println("I found " + p.getName());
   }
}
```

- The Map’s **`get()`** function searches the map for the **key** that matches according to the **`equals()`** method defined for the key’s class.
- The appropriate **value** object is returned.
- **`HashCode()`** values are used to make it fast.

Output:

```
> java map
I have 3 contacts
I found Frank Sinatra
```

CSE1030 – Lecture #7

- Review
- Theory: “is-a” versus “has-a”
- Special Case 1: Has 1
- Special Case 2: Has a “Known” Number
- General Case: Collections
- Retrieving Data from a Collection
- We’re Done!

Next topic…

Aggregation and Composition II