Collections of Objects

Object Oriented Programming
2016375 - 5

Camilo López

Outline

- What are Collections?
- Arrays as Simple Collections
- ArrayList
- HashMap
- TreeMap
- Simultaneous References
- Revisiting the Student Class Design
- Inventing Our Own Collection Types

What are Collections?

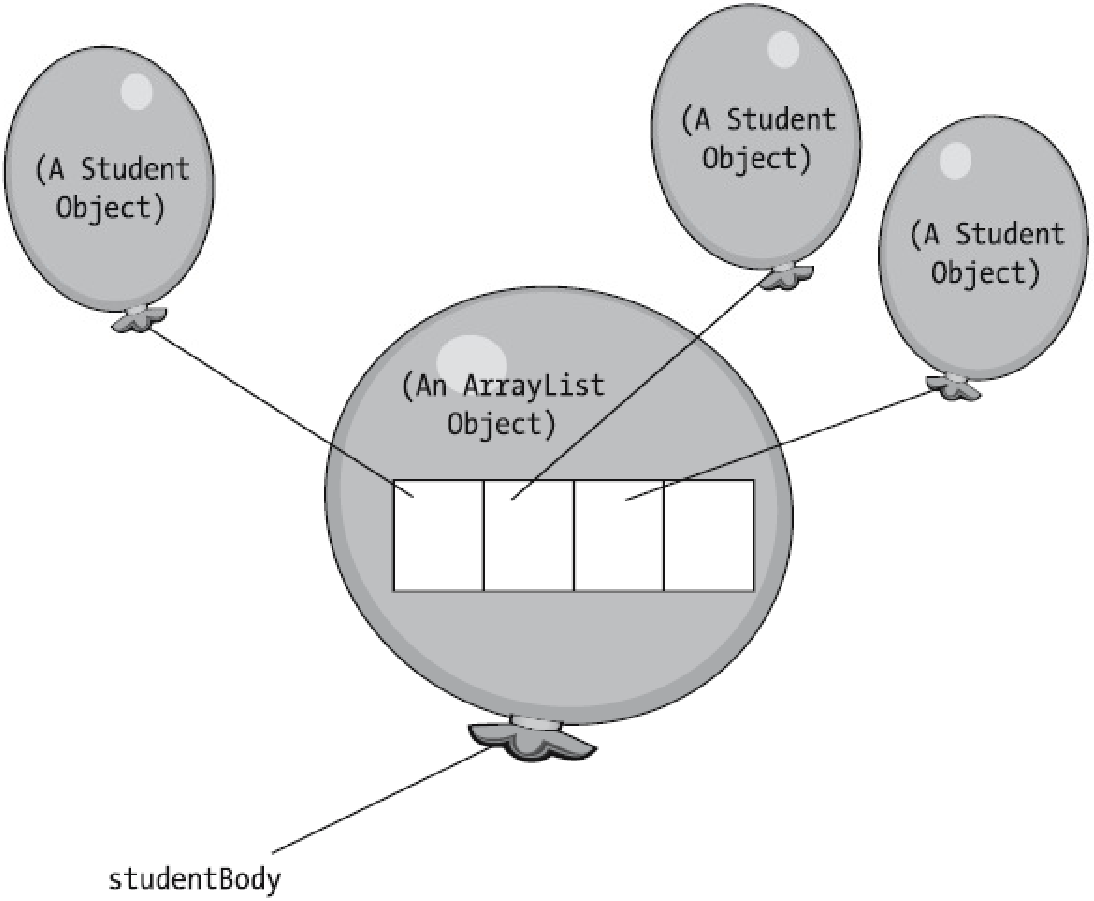
- A ***collection*** — sometimes called a container — is simply an object that groups multiple elements into a single unit.

- Collections are used to store, retrieve, manipulate, and communicate aggregate data.

- Typically, they represent ***data items that form a natural group***, such as a poker hand (a collection of cards), a mail folder (a collection of letters), or a telephone directory (a mapping of names to phone numbers).

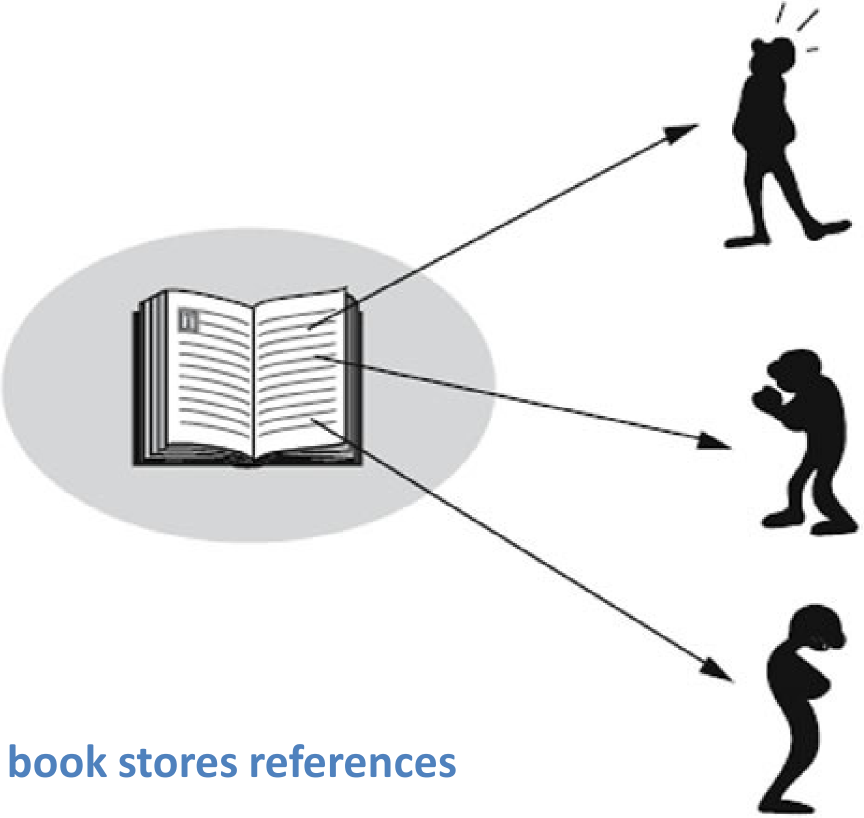
What are Collections?

Organize References to Other Objects



What are Collections?

Organize References to Other Objects

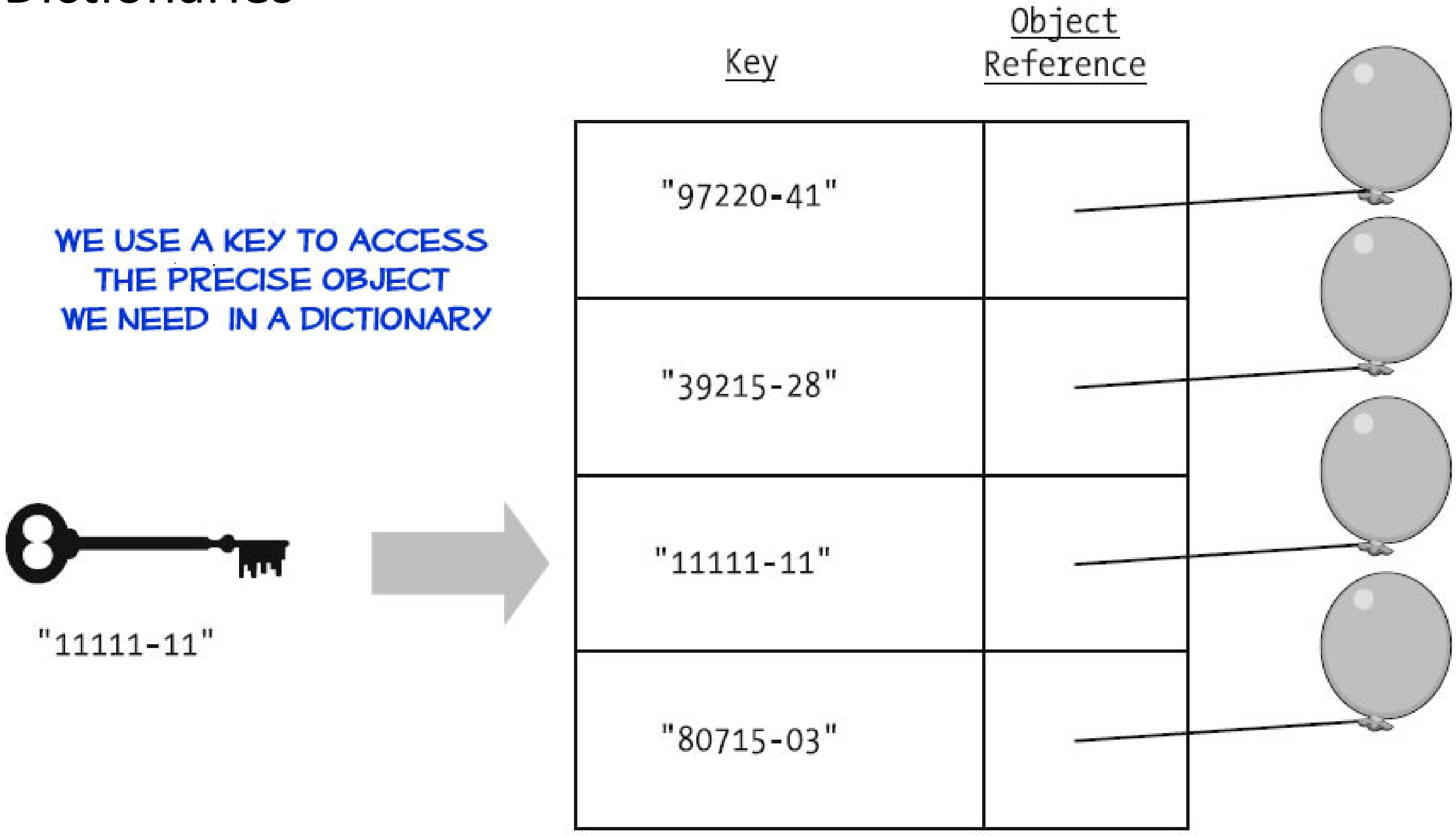


An Address book stores references

What are Collections?

Generic Types of Collections

- Dictionaries



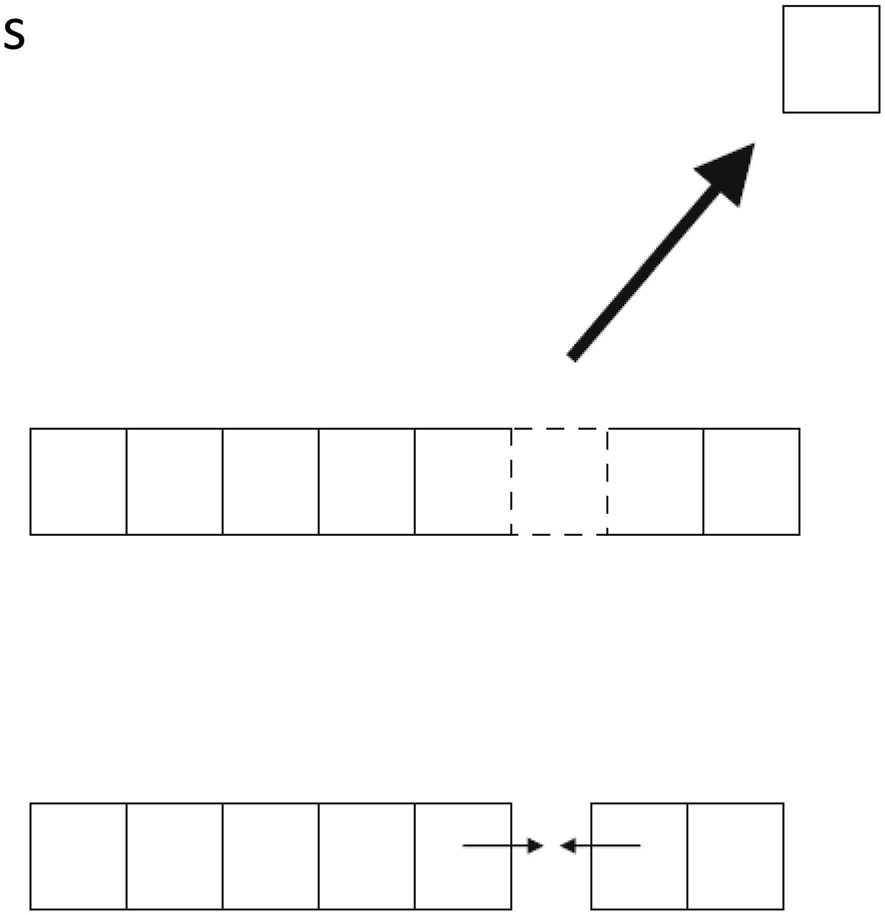
WE USE A KEY TO ACCESS THE PRECISE OBJECT WE NEED IN A DICTIONARY

Key	Object Reference
"97220-41"	
"39215-28"	
"11111-11"	
"80715-03"	

What are Collections?

Generic Types of Collections

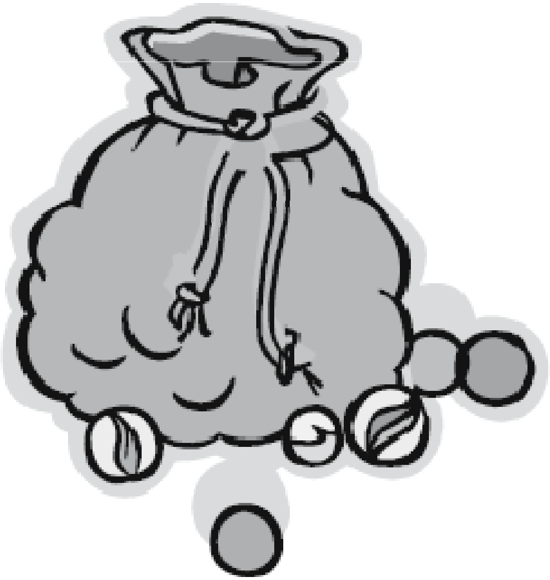
- Ordered Lists



What are Collections?

Generic Types of Collections

- Sets
  - It models the mathematical set abstraction.
  - Duplicate entries aren't allowed in a set.



What are Collections?

Collections Are Encapsulated

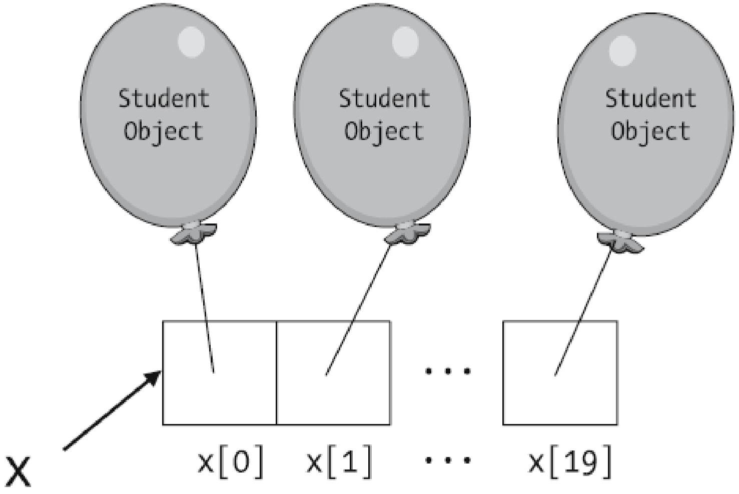
- We don’t need to know the private details of how object references are stored internally to a specific type of collection.
- Public features, such as:
  - Adding objects
  - Removing objects
  - Retrieving specific individual objects
  - Iterating through the objects in some predetermined order
  - Getting a count of the number of objects presently referenced by the collection
  - Answering a true/false question as to whether a particular object’s reference is in the collection or not

Arrays as Simple Collections

- Declaring arrays

dataType[] arrayName = new dataType[size];

```
int[] x = new int[5];
int[] y = new int{1,2,3,4,5};
Student[] x = new Student[20];
Object y = Array.newInstance(Class.forName("Student"), 20);
```



Arrays as Simple Collections

Manipulating Arrays of Objects

- Declaring arrays

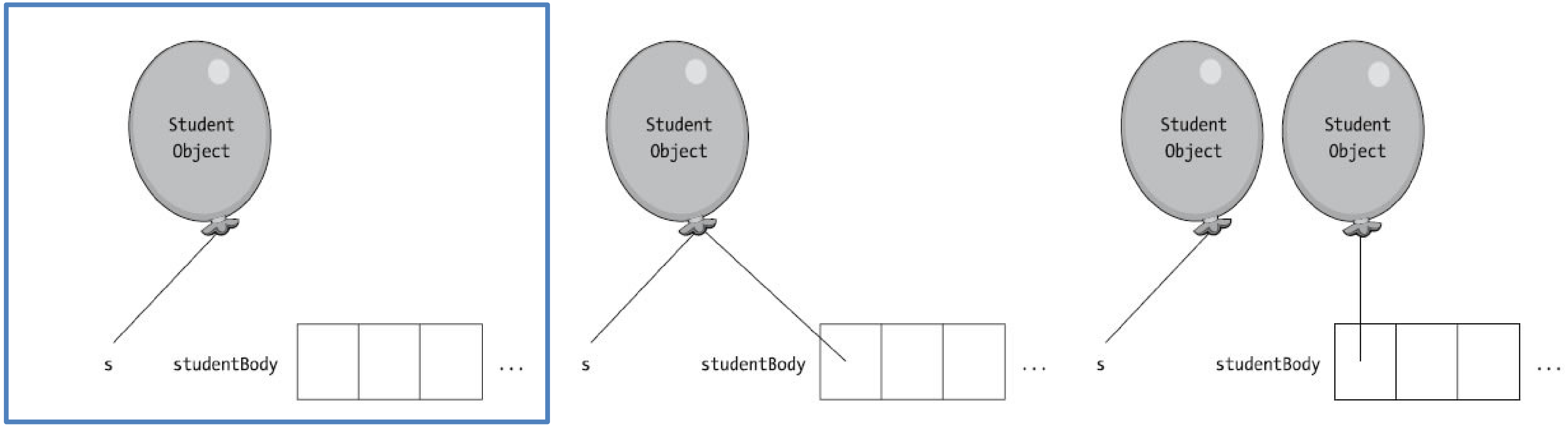
```
dataType[] arrayName = new dataType[size];
```

```
Student[] studentBody = new Student[100];
studentBody[0] = new Student("Fred");
studentBody[1] = new Student("Mary");
// etc.
```

```
Student[] studentBody = new Student[100];
Student s = new Student("Fred");
studentBody[0] = s;

s = new Student("Mary");
studentBody[1] = s;
// etc.
```

Arrays as Simple Collections

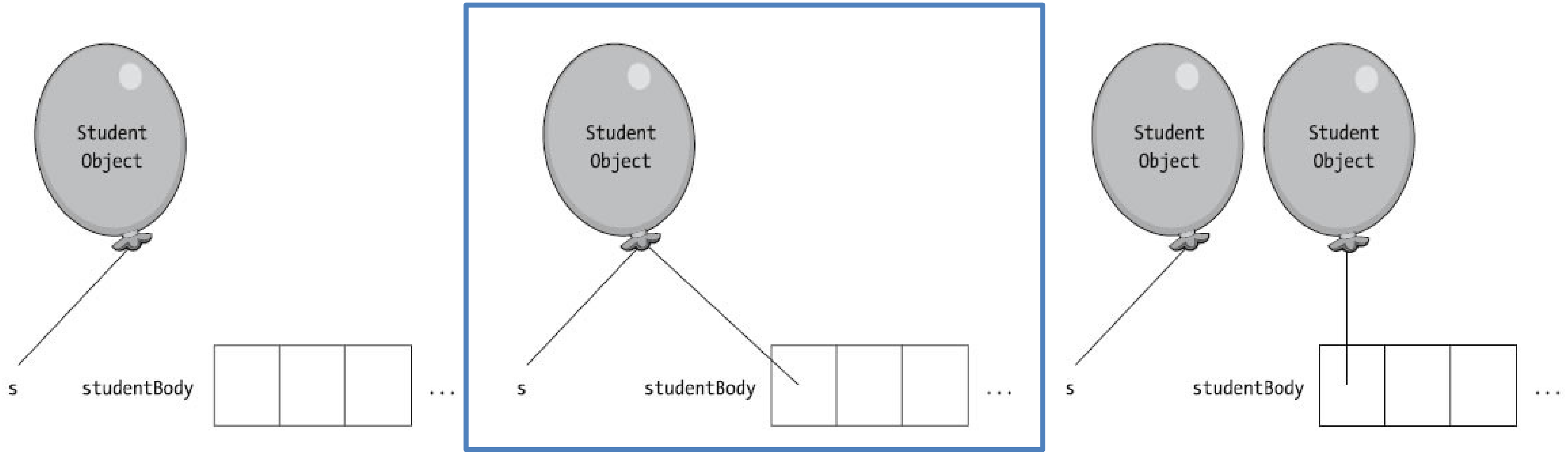
Manipulating Arrays of Objects



```
Student[] studentBody = new Student[100];
Student s = new Student("Fred");
studentBody[0] = s;

s = new Student("Mary");
studentBody[1] = s;
// etc.
```

Arrays as Simple Collections

Manipulating Arrays of Objects

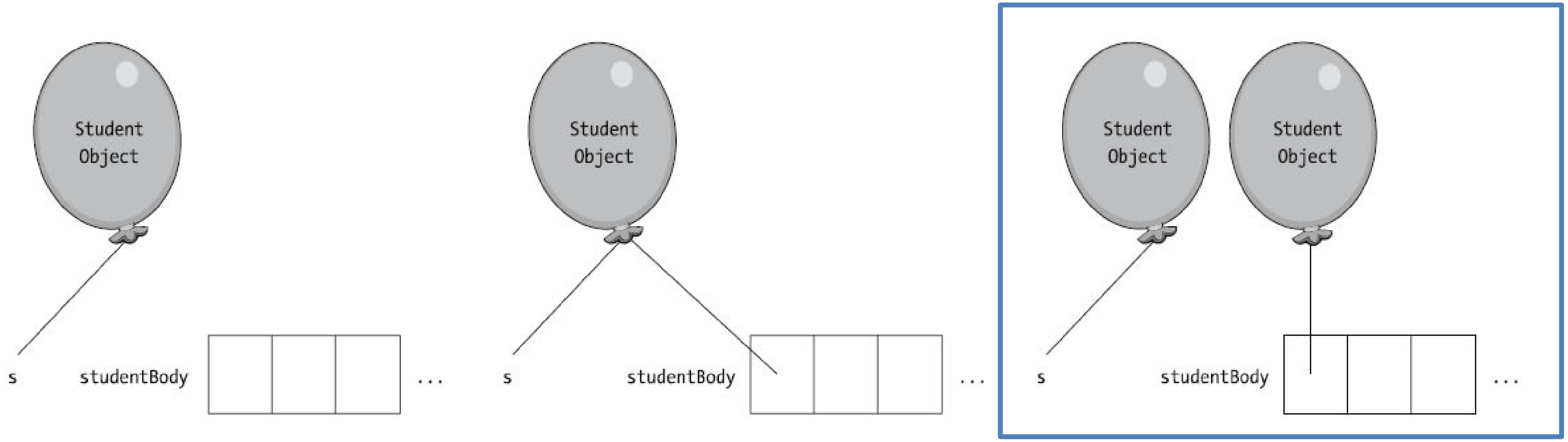


```
Student[] studentBody = new Student[100];
Student s = new Student("Fred");
studentBody[0] = s;

 s = new Student("Mary");

System.out.println(s.getName());
System.out.println(studentBody[0].getName());
```

Both print: Fred

Arrays as Simple Collections

Manipulating Arrays of Objects



```
Student[] studentBody = new Student[100];
Student s = new Student("Fred");
studentBody[0] = s;

 s = new Student("Mary");

System.out.println(s.getName());
System.out.println(studentBody[0].getName());
```

prints: Mary
prints: Fred

ArrayList

- One of the most commonly used predefined Java collection classes.

ArrayList<*ObjectType*> arrayListName = new ArrayList<*ObjectType*>();

```
ArrayList<Professor> faculty = new ArrayList<Professor>();
ArrayList<String> names = new ArrayList<String>();
```

For more info:

http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html

ArrayList

Features

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- boolean add(Object element):

```
students.add(s);
students.add(new Student());

students.add(new GraduateStudent());
students.add(new Person());
```

This is WRONG

- void add(int index, Object element):

```
students.add(3, s);
students.add(0,new Student());

students.add(1,new GraduateStudent());
```

ArrayList

Features

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- boolean addAll(Collection<? extends E> c):

Same type or subtype

```
CollectionType<GradStudent> x = new CollectionType<GradStudent>();

x.add(new GradStudent("John"));
x.add(new GradStudent("Mary"));

students.add(new GradStudent("Holden"));
students.add(new GradStudent("Caulfield"));

students.addAll(x);
```

ArrayList

Features

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- void clear()
- int size()
- boolean isEmpty()
- boolean remove(Object element):

```
if (students.isEmpty()){
    students.add(s);
}else {
    student.remove(s);
}

students.clear();
```

ArrayList

Features

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- boolean contains(Object element)

```
Student s2 = new Student();
students.add(s);

Student s3 = s;

if (students.contains(s2)) { ... }

if (students.contains(s3)) { ... }
```

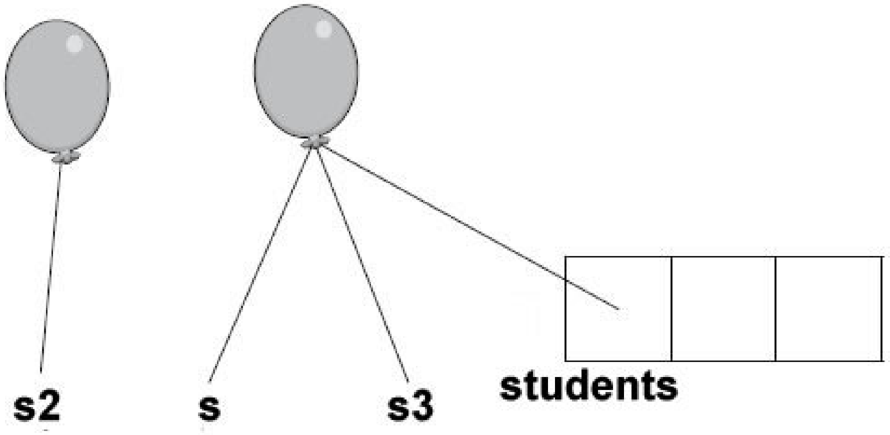
ArrayList

Features

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- boolean contains(Object element)

```
Student s2 = new Student();
students.add(s);

Student s3 = s;

if (students.contains(s2)) { ... }

if (students.contains(s3)) { ... }
```



ArrayList

Iterating Through ArrayLists

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- for (*type referenceVariable* : *CollectionName*) {

  //Do whatever you need

  }

```
for (Student s : students) {
    System.out.println(s.getName());
}
```

ArrayList

Copying the Contents of an ArrayList into an Array

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- *type*[] toArray(*type*[] arrayRef)

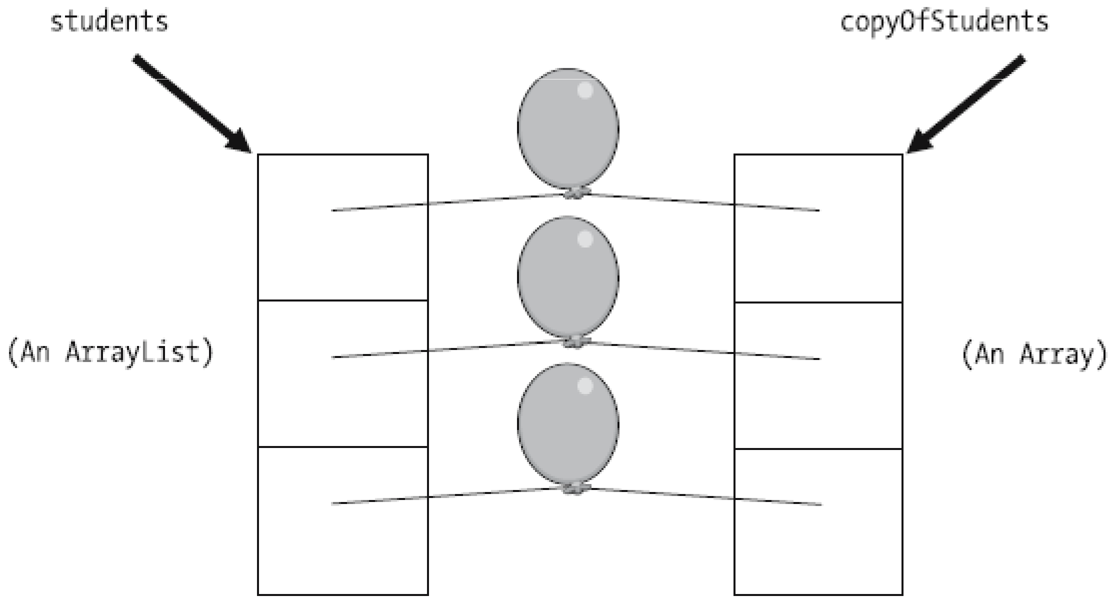
```
students.add(new Student("Herbie"));
students.add(new Student("Klemmie"));
students.add(new Student("James"));

Student[] copyOfStudents = new Student[students.size()];

students.toArray(copyOfStudents);
```

ArrayList

Copying the Contents of an ArrayList into an Array

```
ArrayList<Student> students = new ArrayList<Student>();
Student s = new Student();
```

- *type*[] toArray(*type*[] arrayRef)



ArrayList

```
import java.util.ArrayList;

public class ArrayListExample {
    public static void main (String[] args) {

        ArrayList<Student> students = new ArrayList<Student>();

        Student a = new Student("Herbie");
        Student b = new Student("Klem");

        students.add(a);
        students.add(b);

        for (Student s : students) {
            System.out.println(s.getName());
        }
    }
}
```

HashMap

- A dictionary type collection

Also an Object

```
HashMap<keyType, ObjectType> hashMapName =
                    new HashMap<keyType, ObjectType>();
```

```
HashMap<String, Student> students = new HashMap<String, Student>();
HashMap<String, String> names = new HashMap<String, String>();
```

For more info:

http://java.sun.com/javase/6/docs/api/java/util/HashMap.html

HashMap

Features

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- ObjectType put(Object key, Object value):

```
students.put(s.getId(),s);
students.put(“251234”, new Student(“251234”,John Doe));

students.add(“250099”, new GraduateStudent(“250099”, “Mary”));
students.put(“251234”, s);
```

HashMap

Features

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- ObjectType put(Object key, Object value):

```
students.put(s.getId(),s);
students.put(“251234”, new Student(“251234”,John Doe));

students.add(“250099”, new GraduateStudent(“250099”, “Mary”));
students.put(“251234”, s);
```

A student will be silently replaced!

```
if (!(students.containsKey(“251234”))) {
    students.put(“251234”, s);
}else {
    System.out.println("ERROR: Duplicate student ID found in HashMap: " +
                                  s1.getIdNo());
}
```

HashMap

Features

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- ObjectType get(Object key):

```
Student p = students.get(s.getId());

System.out.println(p.getName());

System.out.println(students.get(s.getId()).getName());

System.out.println(students.get(“272671”).getName());
```

All print: Juan Perez

HashMap

Features

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- void clear()
- int size()
- boolean isEmpty()
- boolean remove(Object key)
- boolean containsKey(Object key)
- boolean containsValue(Object value)

HashMap

Iterating Through HashMaps

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- for (*type referenceVariable* : *CollectionName*) {

  //Do whatever you need

  }

```
for (Student s : students.values()) {
    System.out.print(s.getName());
}
```

HashMap

Iterating Through HashMaps

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

- for (*type referenceVariable* : *CollectionName*) {

  //Do whatever you need

  }

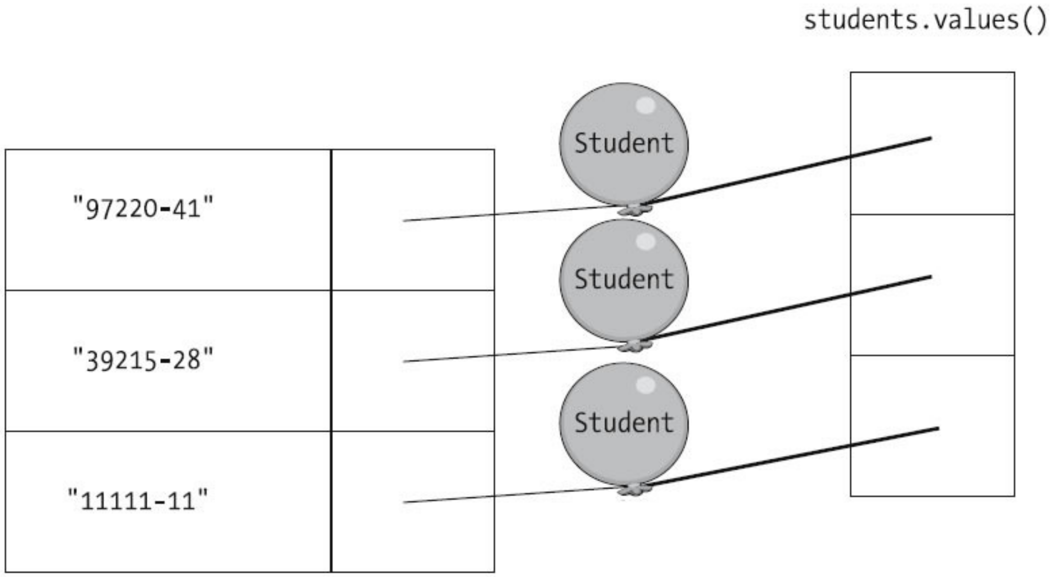
```
for (Student s : students.values()) {
    System.out.print(s.getName());
}
```

HashMap

Iterating Through HashMaps

```
HashMap<String, Student> students = new HashMap<String, Student>();
Student s = new Student(“272671”,”Juan Perez”);
```

```
for (Student s : students.values()) {
    System.out.print(s.getName());
}
```



the values() method returns a collection view of the values contained in this map

the keySet() method returns a set view of the keys contained in this map

TreeMap

- Another dictionary type collection

Also an Object

TreeMap<*keyType, ObjectType*> treeMapName =

new TreeMap<*keyType, ObjectType*>();

```
TreeMap<String, Student> students = new TreeMap<String, Student>();
TreeMap<String, String> names = new TreeMap<String, String>();
```

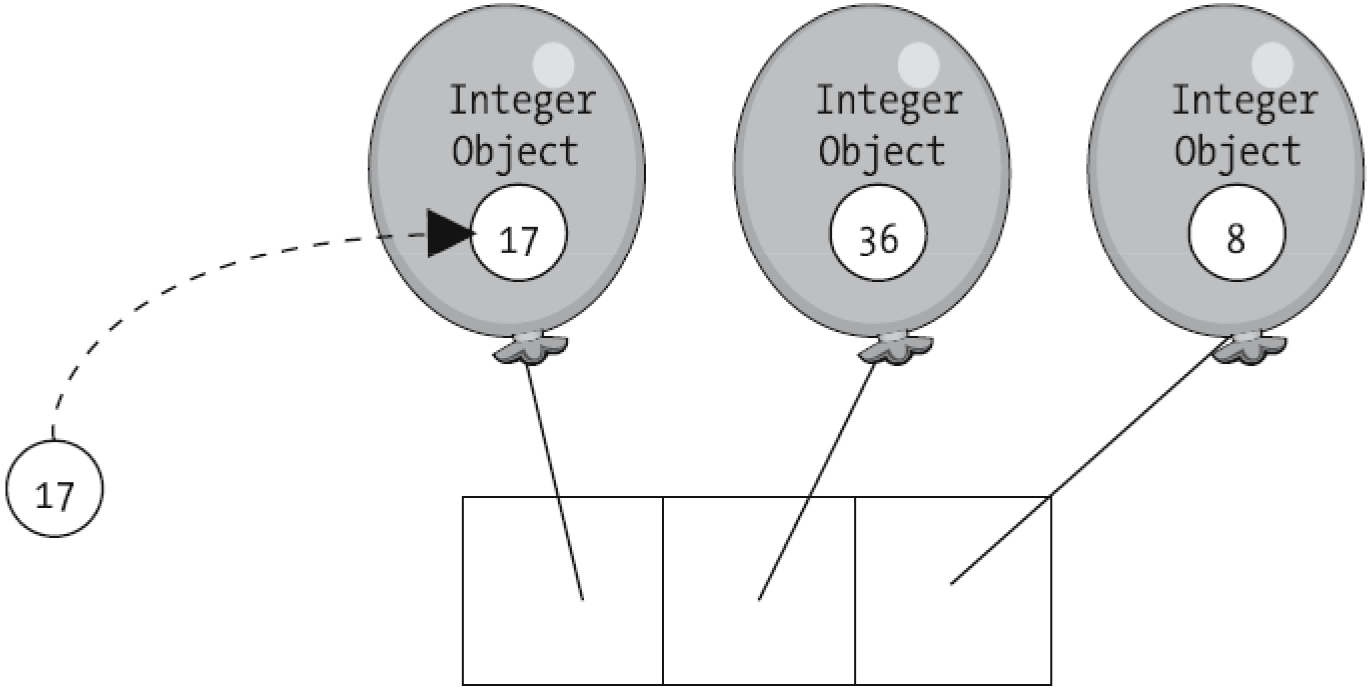
For more info:

http://java.sun.com/j2se/1.4.2/docs/api/java/util/TreeMap.html

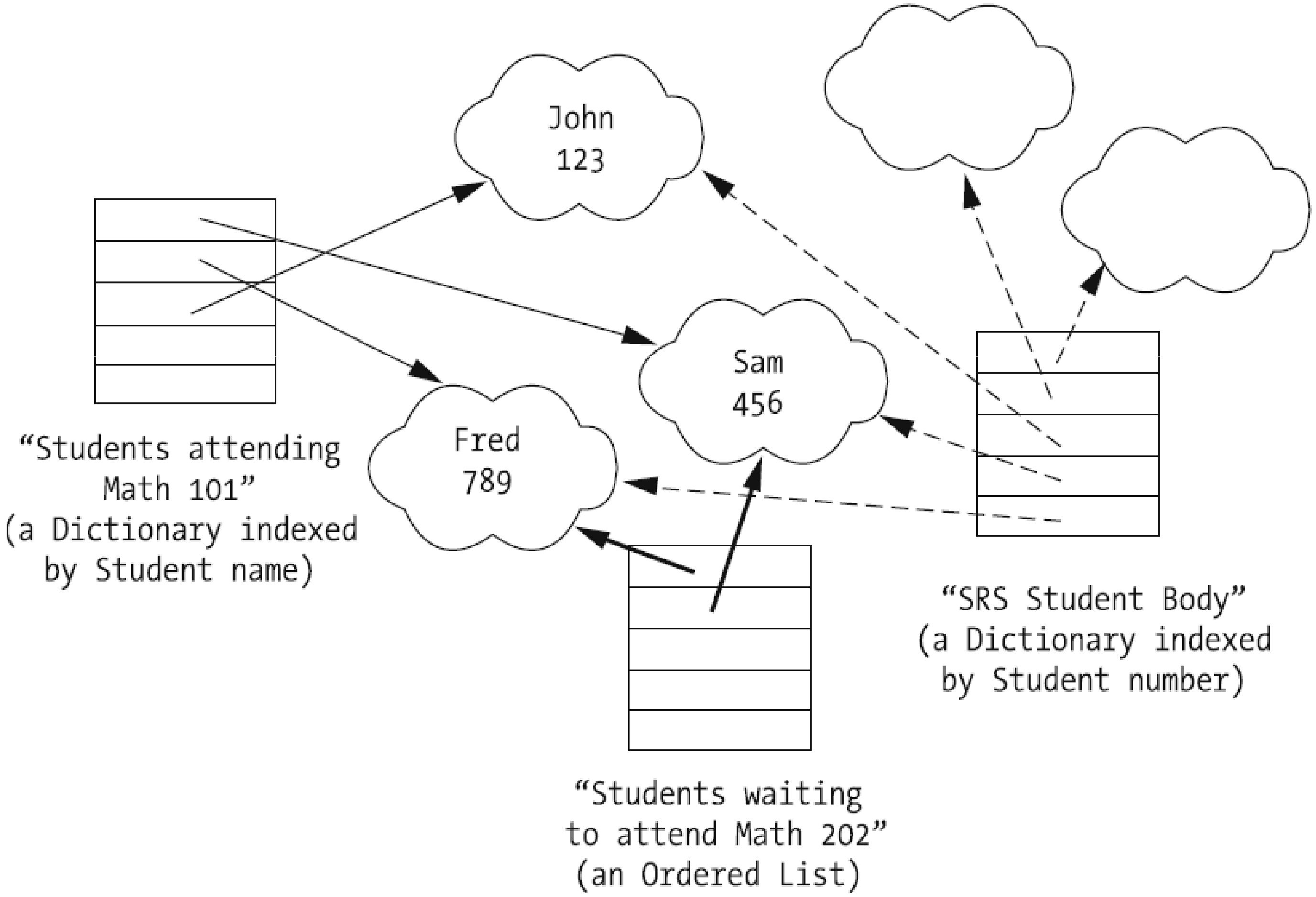
TreeMap

- As another dictionary type collection, TreeMaps are very similar to HashMaps, with one notable difference:
  - When we iterate through a TreeMap, objects are automatically retrieved from the collection in ascending key (sorted) order.
  - When we iterate through a HashMap, on the other hand, there’s no guarantee as to the order in which items will be retrieved.

Wrapper Classes for Primitive Types



Simultaneous References



"Students attending Math 101" (a Dictionary indexed by Student name)

"Students waiting to attend Math 202" (an Ordered List)

"SRS Student Body" (a Dictionary indexed by Student number)

Inventing Our Own Collection Types

- ***Approach #1:*** We can design a brand-new collection class from scratch.

- ***Approach #2:*** We can use the techniques that we learned when we saw the inheritance to extend a predefined collection class.

- ***Approach #3:*** We can create a "wrapper" class that encapsulates one of the built-in collection types, to "abstract away" some of the details involved with manipulating the collection.

Inventing Our Own Collection Types

*Extending a Predefined Collection Class (*MyIntCollection*)*

```
public class MyIntCollection extends ArrayList<Integer> {
    private int smallestInt;
    private int largestInt;
    private int total;

    public MyIntCollection() {
        super();
        total = 0;
    }

    public int getSmallestInt() {
        return smallestInt;
    }

    public int getLargestInt() {
        return largestInt;
    }
//cont…
```

Inventing Our Own Collection Types

*Extending a Predefined Collection Class (*MyIntCollection*)*

```
//cont…

 public boolean add(int i) {
        if (this.isEmpty()) {
            smallestInt = i;
            largestInt = i;
        }else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return super.add(i);
    }

    public double getAverage() {
        return ((double) total) / ((double) this.size());
    }
}
```

Inventing Our Own Collection Types

*(*MyIntCollection*) Client Code*

```
public class MyIntCollectionExample {
    public static void main(String[] args) {
        MyIntCollection mic = new MyIntCollection();
        mic.add(3);
        mic.add(6);
        mic.add(1);
        mic.add(9);

        System.out.println("The collection contains " + mic.size() + " int values");

        System.out.println("The smallest value is: " + mic.getSmallestInt());
        System.out.println("The largest value is: " + mic.getLargestInt());
        System.out.println("The average is: " + mic.getAverage());
    }
}
```

Inventing Our Own Collection Types

Encapsulating a Standard Collection (MyIntCollection2)

```
public class MyIntCollection2 {
    ArrayList<Integer> numbers;
    private int smallestInt;
    private int largestInt;
    private int total;

    public MyIntCollection2() {
        numbers = new ArrayList<Integer>();
        total = 0;
    }

    public int getSmallestInt() {
        return smallestInt;
    }

    //cont…
```

Inventing Our Own Collection Types

*Encapsulating a Standard Collection (*MyIntCollection2*)*

```
//…cont

public int getLargestInt() {
    return largestInt;
}

public int size() {
    return numbers.size();
}

public double getAverage() {
    return ((double) total)/this.size();
}

//cont…
```

Inventing Our Own Collection Types

*Encapsulating a Standard Collection (*MyIntCollection2*)*

```
    //…cont

    public boolean add(int i) {
        if (numbers.isEmpty()) {
            smallestInt = i;
            largestInt = i;
        } else {
            if (i < smallestInt) smallestInt = i;
            if (i > largestInt) largestInt = i;
        }
        total = total + i;
        return numbers.add(i);
    }
}
```

Inventing Our Own Collection Types

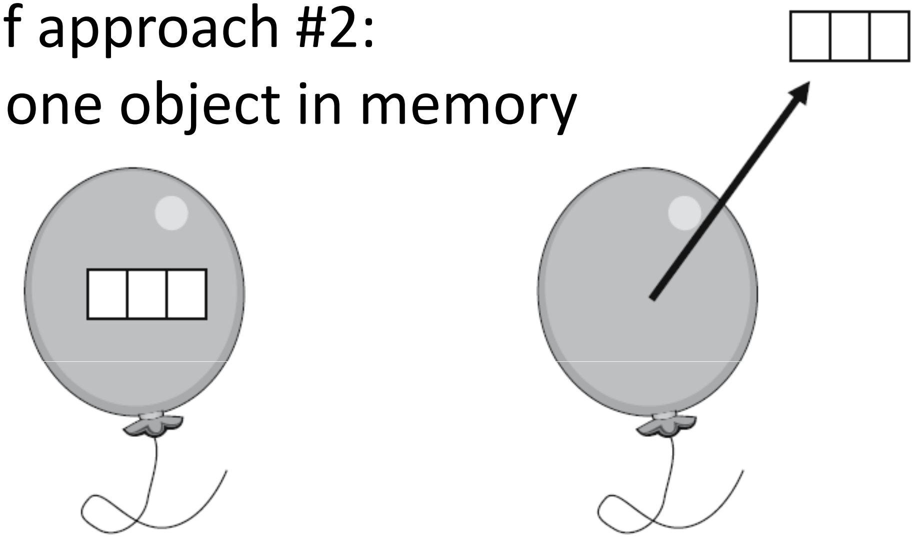
*(*MyIntCollection2*) Client Code*

```
public class MyIntCollection2Example {
    public static void main(String[] args) {
        MyIntCollection2 mic = new MyIntCollection2();
            mic.add(3);
            mic.add(6);
            mic.add(1);
            mic.add(9);

        System.out.println("The collection contains " + mic.size() + " int values");

        System.out.println("The smallest value is: " + mic.getSmallestInt());
        System.out.println("The largest value is: " + mic.getLargestInt());
        System.out.println("The average is: " + mic.getAverage());
    }
}
```

Inventing Our Own Collection Types

Trade-offs of Approach #2 vs. Approach #3

- An advantage of approach #2:
  We create only one object in memory

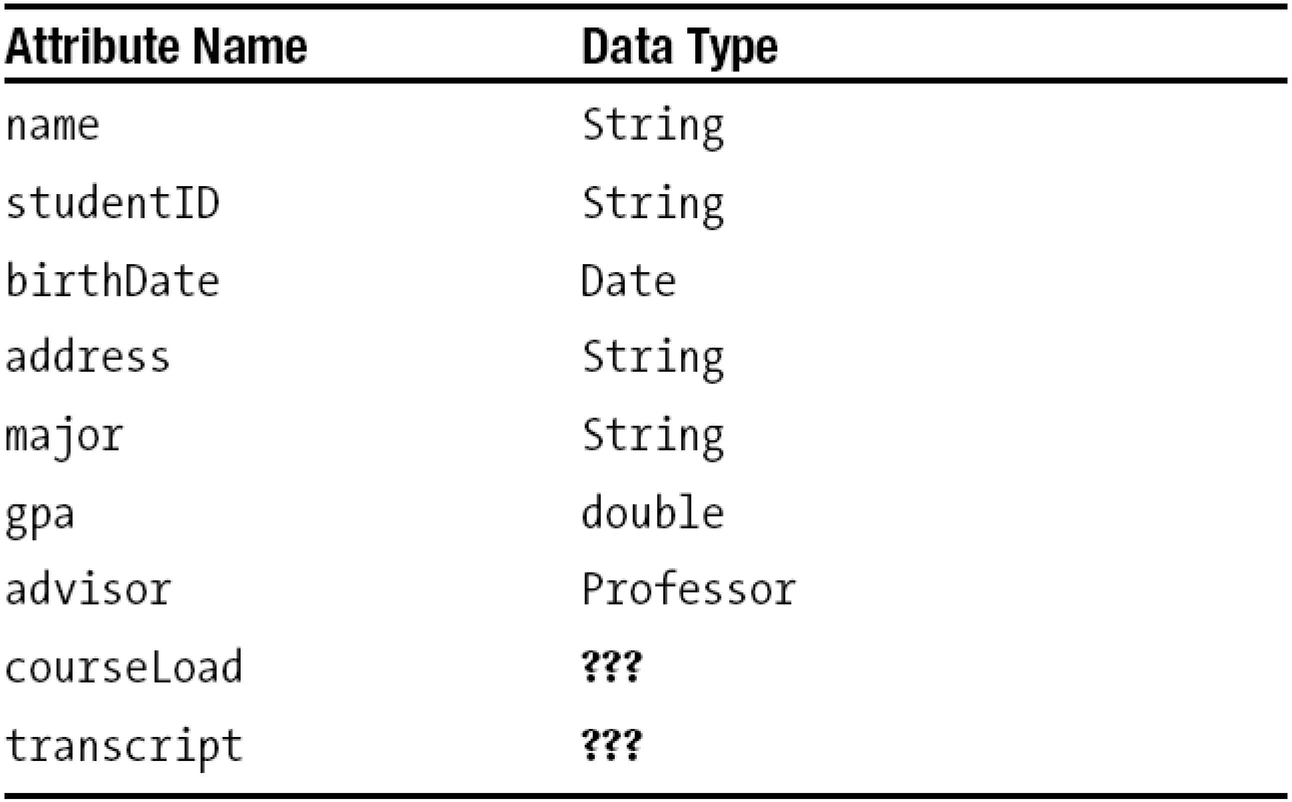


An instance of MyIntCollection ***is*** an ArrayList (***one*** object instance), whereas ...

An instance of MyIntCollection2 ***refers to*** an ArrayList (***two*** object instances)

- An advantage of approach #3:
  Encapsulation → Information Hiding

Revisiting the Student Class Design



Attribute Name	Data Type
name	String
studentID	String
birthDate	Date
address	String
major	String
gpa	double
advisor	Professor
courseLoad	**???**
transcript	**???**

Revisiting the Student Class Design

- courseLoad:
  Represents a list of all Course objects that the Student is presently enrolled in

- transcript:
  It's a report of all of the courses that a has taken, along with the semester in which each course was taken, its number of credit hours, and the letter grade that the student received. A typical transcript entry, when printed, might look as follows:

  CS101   Beginning Objects   3.0   A

Revisiting the Student Class Design

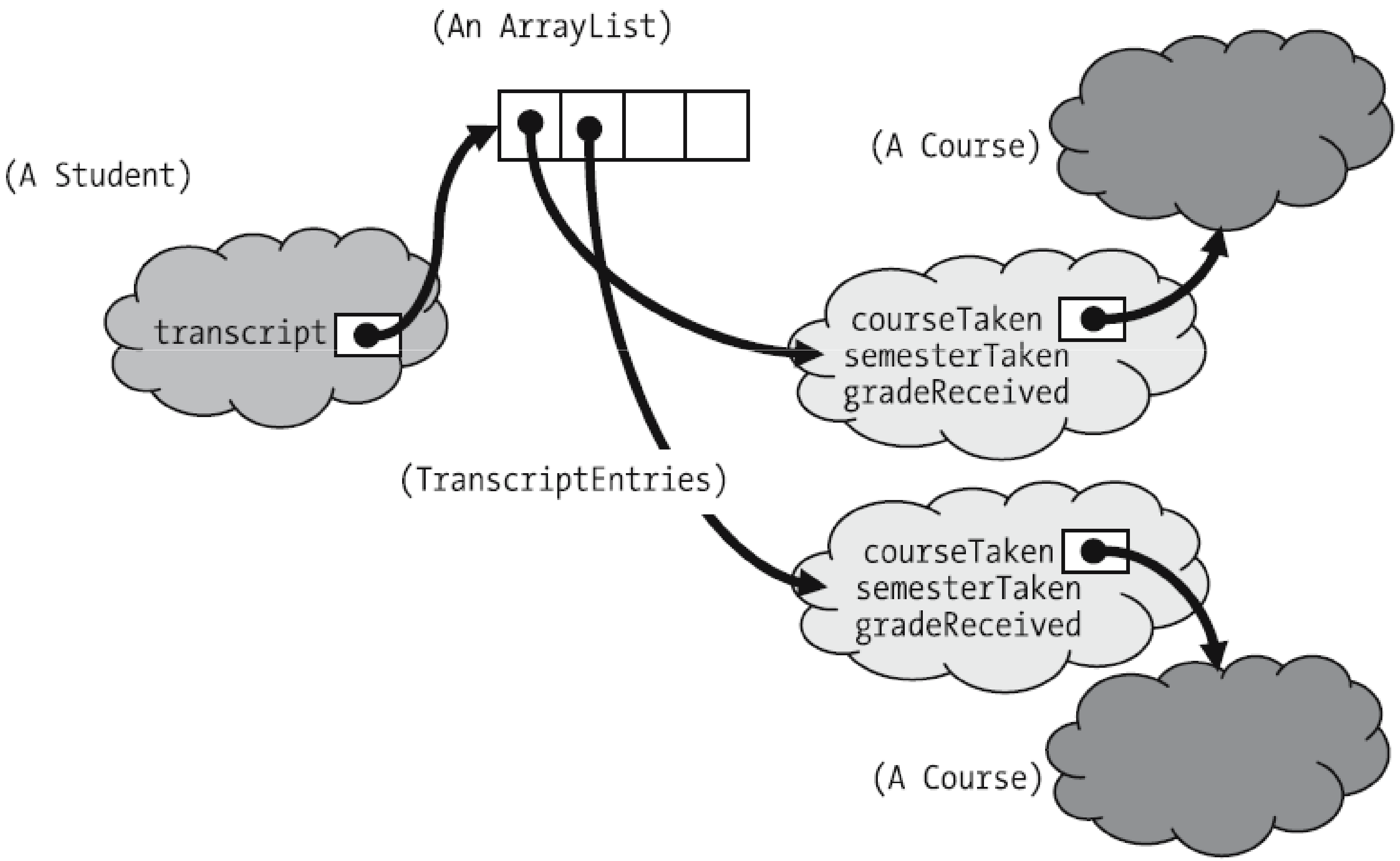
- courseLoad:

```
    private ArrayList<Course> courseLoad;
```

- transcript:

```
public class TranscriptEntry {
    private Course courseTaken;
    private String semesterTaken;
    private String gradeReceived;

    public TranscriptEntry(Course c, String semester, String grade) {…}

    public void printTranscriptEntry() {
        System.out.println(this.getCourseTaken().getCourseNo() + "\t" +
            this.getCourseTaken().getTitle() + "\t" +
            this.getCourseTaken().getCreditHours() + "\t" +
            this.getGradeReceived());
    }
}
```

Revisiting the Student Class Design



Revisiting the Student Class Design

Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
TranscriptEntry te = new TranscriptEntry(c, “2009-I", "A+");
s.addTranscriptEntry(te);

// Additional grades assigned for other courses ... details omitted.

s.printTranscript();
```

It’s not very intuitive for someone reading this client code

Revisiting the Student Class Design

Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
TranscriptEntry te = new TranscriptEntry(c, “2009-I", "A+");
s.addTranscriptEntry(te);



s.printTranscript();
```

```
public class Student {
    private ArrayList<TranscriptEntry> transcript;

    public void addTranscriptEntry(TranscriptEntry te) {
        transcript.add(te);
    }
}
```

Client code has to be aware of the notion of a TranscriptEntry.

.

Revisiting the Student Class Design

Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
TranscriptEntry te = new TranscriptEntry(c, “2009-I", "A+");
s.addTranscriptEntry(te);



s.printTranscript();
```

```
public class Student {
    private ArrayList<TranscriptEntry> transcript;

    public void printTranscript() {
        for (TranscriptEntry te : transcript) {
            te.printTranscriptEntry();
        }
    }
}
```

Revisiting the Student Class Design

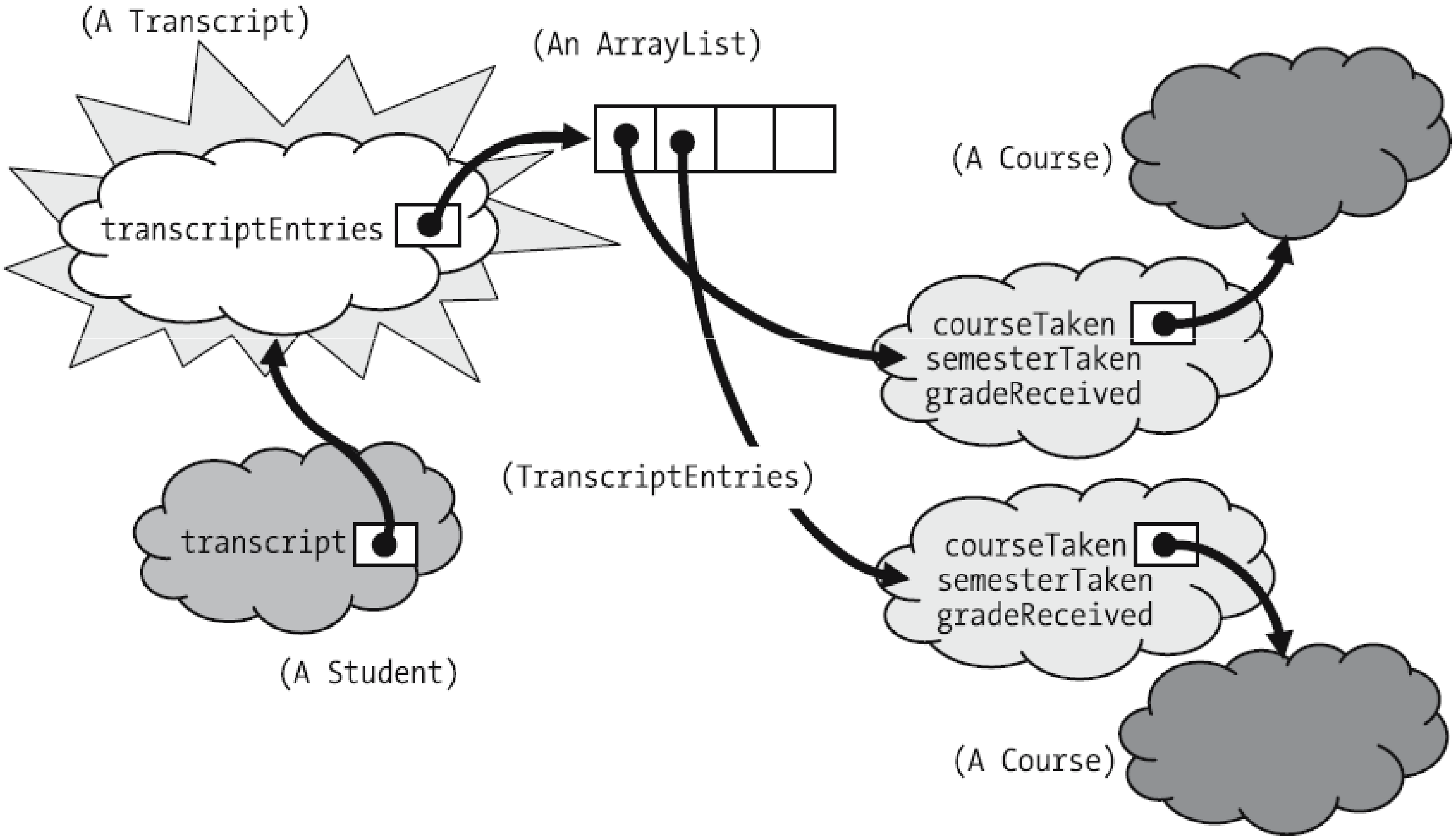
A different approach

Encapsulates an ArrayList of TranscriptEntry references.

```
public class Transcript {
    private ArrayList<TranscriptEntry> transcriptEntries;
    Student owner;

    // Constructor/accessor details omitted.

    public void courseCompleted(Course c, String semester, String grade) {
        transcriptEntries.add(new TranscriptEntry(c, semester, grade);
    }

    public void print() {
        for (TranscriptEntry te : transcript) {
            te.printTranscriptEntry();
        }
    }
// etc.
}
```

Maintain a handle on the Student owner of this transcript

Revisiting the Student Class Design



Revisiting the Student Class Design

Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
s.courseCompleted(c, “2009-I", "A+");


// Additional grades assigned for other courses ... details omitted.

s.printTranscript();
```

Revisiting the Student Class Design

Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
s.courseCompleted(c, “2009-I", "A+");



s.printTranscript();
```

```
public class Student {
    private Transcript transcript;

    public void courseCompleted(Course c, String
        semester, String grade) {

        Transcript.courseCompleted(c, semester, grade);
    }
}
```

This method hides more “gory details,” and is hence easier for client code to use.

Revisiting the Student Class Design

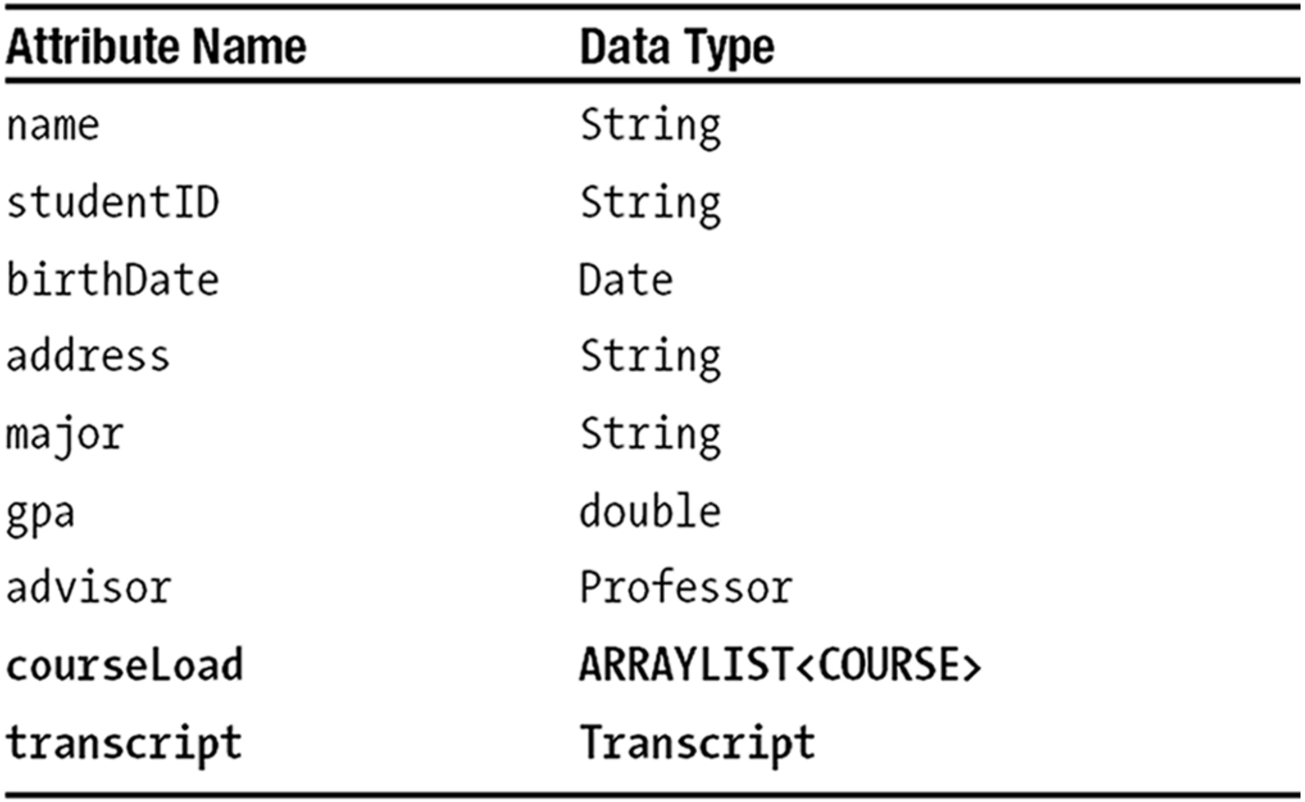
Client Code

```
Student s = new Student("1234567", "John Doe");
Course c = new Course("LANG 800", "Advanced Language Studies");
s.registerForCourse(c);

// Semester is finished! Assign a grade to this student.
s.courseCompleted(c, “2009-I", "A+");



s.printTranscript();
```

```
public class Student {
    private Transcript transcript;

    public void printTranscript() {
        transcript.print()
    }
}
```

Delegation!!

Revisiting the Student Class Design



Attribute Name	Data Type
name	String
studentID	String
birthDate	Date
address	String
major	String
gpa	double
advisor	Professor
courseLoad	**ARRAYLIST<COURSE>**
transcript	**Transcript**

References

- J. Barker, *Beginning Java Objects: From Concepts To Code, Second Edition*, Apress, 2005.
- Java^{TM} Platform, Standard Edition 6 – The Collection Framework. Available online at: http://java.sun.com/javase/6/docs/technotes/guides/collections/index.html