Object Oriented Programming

Debapriyo Majumdar

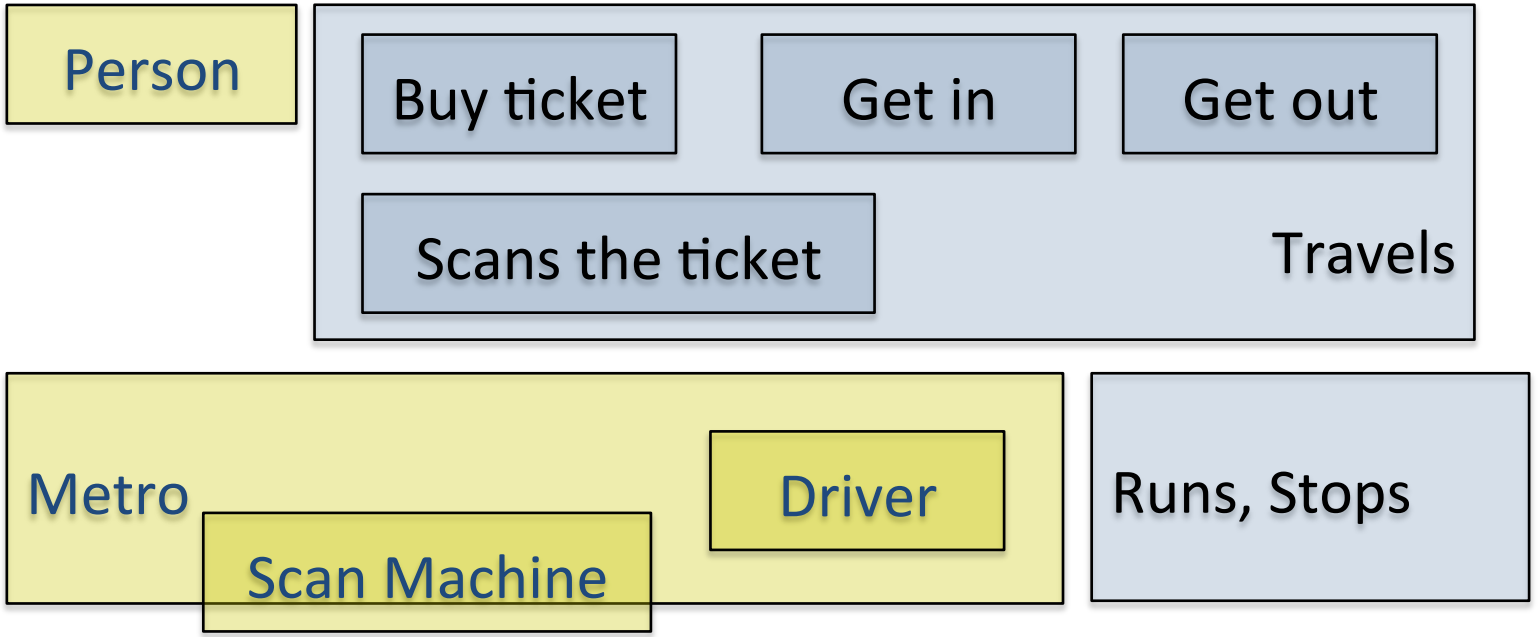
Programming and Data Structure Lab
M Tech CS I – Semester I
Indian Statistical Institute Kolkata

August 7 and 14, 2014

Objects

Real world *objects*, or even people!

A person travels by metro.



- Different *objects* all performing their duties
- Objects are related to each other too

Objects

- State and Behavior

- States of the person
  - Name, Profession, Current location, …
- States of the metro
  - AC/Non-AC, Color, Location, Number of coaches, …

- Behavior: actions performed by the object
  - Person: Travel, Attending a class, Eating
  - Metro: Running, Stopping, …

Independence and Dependency

Behavior that matters to the person	Behavior that matters to the metro
Opens the door	Buys the ticket
Transports	Avails all the facilities
Closes the door	
Announces present and next station	
Provides a system for buying tickets	
Provides a system for validating tickets	

- Consider: some changes in the door opening technology
- The functionality for the passenger will remain the same
  - There will be changes in the internal implementation of the doors

Exercise – The PDS Lab

- Exercise: Consider the scenario of this class – PDS Lab course
- Identify the objects, their states and behaviors, what do the objects provide to each other

The Java Hello World Program

- Create a directory called “isical” inside dayX/java
- Open a file named Main.java inside isical

```
package isical;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

Compile and Run!

- Your path: XYZ/java/isical/
- Go to the directory XYZ/java
- Compile:

```
$ javac isical/Main.java
```

- A file called Main.class should be created
- Run

```
$ java isical/Main
```

Object oriented programming

The instructor to write down the name of all the students

Overview:

- A list of students is passed to the instructor
- Every student can tell their names
- Instructor has a method to write down the names

The Student Object

File: Student.java

```
package isical;

public class Student {
    private String name;
    // The name of the student
    private int rollNumber;
    // The numeric roll number of the student
}
```

The Student Object

File: Student.java

```
package isical;

public class Student {
    private String name;
    // The name of the student
    private int rollNumber;
    // The numeric roll number of the student

    /**
     * Constructs an instance of a student
     * @return
     */
    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }

}
```

Now compile Student.java

The Student Object

File: Student.java

```
package isical;

public class Student {
    private String name;
    // The name of the student
    private int rollNumber;
    // The numeric roll number of the student

    /**
     * Constructs an instance of a student
     * @return
     */
    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }

}
```

What does the Student class do for me?

Nothing yet!!

The student needs to tell his/her name and roll number!!

The Student Object

File: Student.java

```
package isical;

public class Student {
    ..........
    ADD THE FOLLOWING LINES
    ..........
    /* What can the student do? */
    public String getName() {
        return name; // Returns the name
    }

    public int getRollNumber() {
        return rollNumber;
    }

}
```

Now (s)he can

The Instructor Object

Filename: ?

```
package isical;

public class Instructor {

     private String name; // The name of the student
     private String unit; // Which unit does the instructor belong to?

     public Instructor(String name, String unit) {
          this.name = name;
          this.unit = unit;
     }

     /* What can the student do? */
     public String getName() {
          return name; // Returns the name
     }

     public String getUnit() {
          return unit; // Returns the name of the unit
     }

}
```

What does (s)he need to do?

Get names from students and write down

The method to do that work

```
public void writeNamesOfStudents(Student[] listOfStudents) {
        for (int index = 0; index < listOfStudents.length;
        index++) {
            String nameOfStudent = listOfStudents[index].getName();
            System.out.println(nameOfStudent);
        }
}
```

The final code

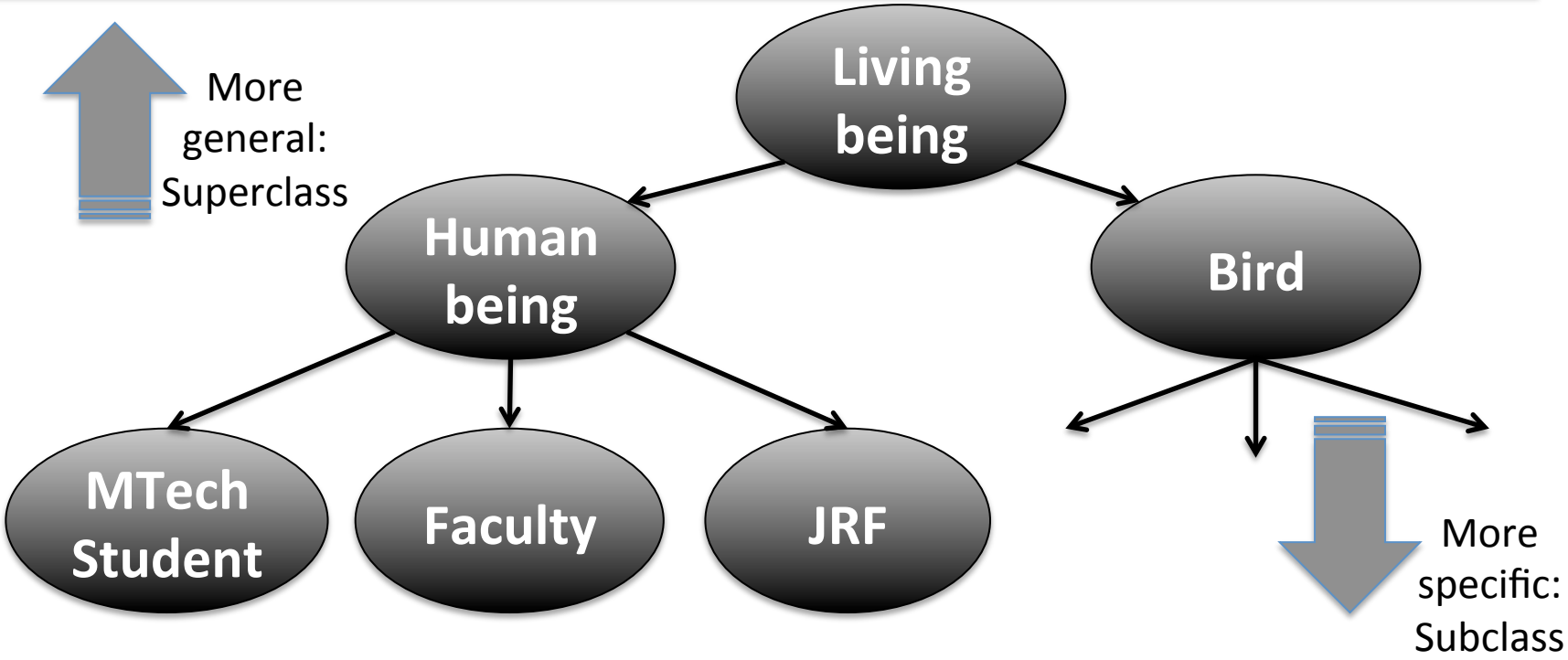
Filename: Main.java

```
package isical;

public class Main {

	public static void main(String[] args) {
		// Create an array of Students
		Student[] listOfStudents = new Student[10];

		// Create the students
		for (int studentNum = 0; studentNum < 10; studentNum++) {
			Student student = new Student("Student Number"+studentNum, studentNum);
			listOfStudents[studentNum] = student;
		}

		// Create an instructor
		Instructor instructor = new Instructor("Instructor1","Some Unit Who cares");

		// Now get the instructor to do the work
		instructor.writeNamesOfStudents(listOfStudents);

	}

}
```

What have we learnt so far?

- The basic object oriented way of thinking
- Java
  - Basic file organization, packages
  - Class, constructor, methods
  - One example

Inheritance



- Subclasses can use *state* and *behavior* of the superclass
- Java: each class can have one *direct superclass*, each superclass can have *any* number of subclasses

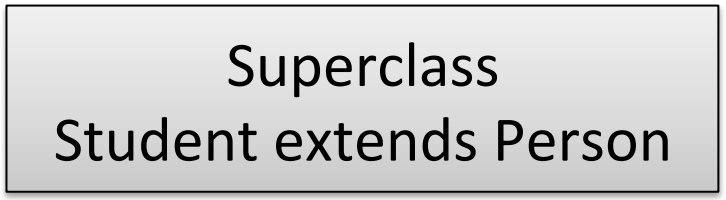
Student, Instructor and Person

```
package isical;

public class Person {

    private String name; // Name of the person
    private int age; // Age of the person

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}
```

Student as a subclass of Person


```
package isical;

public class Student {

    private String name; // The name of the student
    private int rollNumber; // The numeric roll number of the student

    public Student(String name, int age) {
        super(name,age); // Call the constructor of the superclass
    }

    /* What can the student do? */
    public String getName() {
        return name; // Returns the name
    }

    public int getRollNumber() {
        return rollNumber;
    }
}
```

Student as a subclass of Person




```
package isical;

public class Student extends Person {

    private String name; // The name of the student
    private int rollNumber; // The numeric roll number of the student

    public Student(String name, int age) {
        super(name,age); // Call the constructor of the superclass
    }

    /* What can the student do? */
    public String getName() {
        return name; // Returns the name
    }

    public int getRollNumber() {
        return rollNumber;
    }
}
```

Student as a subclass of Person

```
package isical;

public class Student extends Person {

    private String name; // The name of the student
    private int rollNumber; // The numeric roll number of the student

    public Student(String name, int age) {
        super(name,age); // Call the constructor of the superclass
    }

    /* What can the student do? */
    public String getName() {
        return name; // Returns the name
    }

    public int getRollNumber() {
        return rollNumber;
    }
}
```

Do we need it?

Do we need it?

Student as a subclass of Person

```
package isical;

public class Student extends Person {

    private String name; // The name of the student
    private int rollNumber; // The numeric roll number of the student

    public Student(String name, int age) {
        super(name,age); // Call the constructor of the superclass
    }

    /* What can the student do? */
    public String getName() {
        return name; // Returns the name
    }

    public int getRollNumber() {
        return rollNumber;
    }
}
```

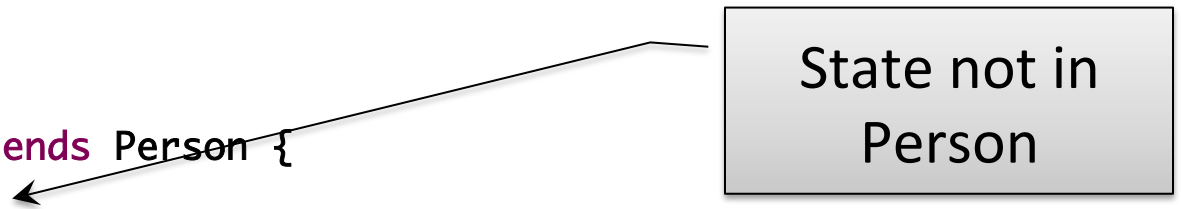
Inherited from Person

Inherited from Person

What else is inherited?

Student as a subclass of Person

```
package isical;

public class Student extends Person {

    private int rollNumber; // The numeric roll number of the student

    public Student(String name, int age) {
        super(name,age); // Call the constructor of the superclass
    }

    /* What can the student do
    (more than what it inherits by being a Person) */
    public int getRollNumber() {
        return rollNumber;
    }
}
```



Now compile, run and verify that a Student can still tell his/her name!

Similarly, modify the code so that the Instructor also extends Person

Overriding a super method

- Now consider the following functionality in Person

```
public class Person {
    …
    …
    private String profession; // Profession of the person
    …
    …
    public void setProfession(String profession) {
          this.profession = profession;
    }

    public String getProfession() {
          return profession;
    }
}
```

The Student would automatically inherit the same!

The Student does it differently!

```
public class Student extends Person {
    …
    …
    private String institute; // The name of the institute
    private String program; // The name of the program
    …
    …

    // Overrides isical.Person.getProfession
    public String getProfession() {
         return "Student of " + program + " at " + institute;
    }
}
```

Abstract class

- *Abstract method*: too *abstract* to provide an implementation
  - Name of a living being
  - Area of a shape (can be circle, rectangle, …)
- *Abstract class*: Some conceptual objects with one or more *abstract methods*
  - Concrete subclasses are *derived* from abstract class

LivingBeing

```
package isical;

public abstract class LivingBeing {

     public abstract String getName();

}
```

Define an abstract class

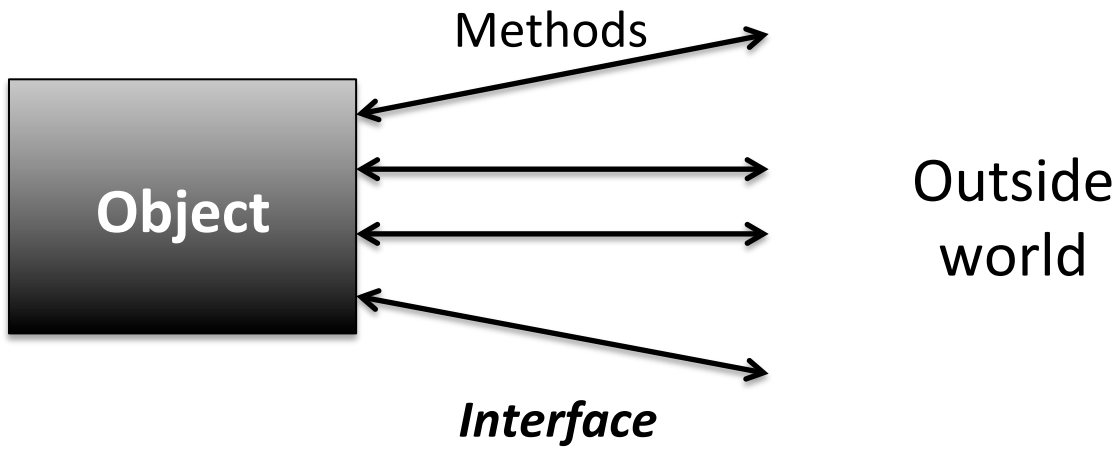
Abstract method

```
public class Person extends LivingBeing {
     … …
     … …
}
```

Derived class

Must implement the inherited abstract method

Interface



- Methods form the object's *interface* with the outside world
- A group of methods to define a set of common functionalities → An interface

An Interface Movable

```
package isical;

public interface Movable {

    public int movePosition(int position);

}
```

Any movable object must have this functionality

```
public class Person extends LivingBeing implements Movable {
    //... ...
    @Override
    public int movePosition(int position) {
        return position+1;
    }

}
```

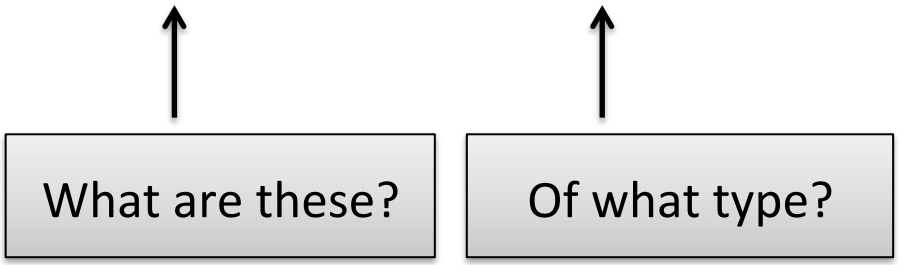
In general a person moves one step

Abstract class vs Interface

- Abstract classes, superclasses and derived classes are *supposed* to be of the same type of objects
  - Living being → Human, Bird
  - House → Bungalow, Hut
- An Interface defines a set of functionalities
  - House: class, Air conditioning: interface
  - Some houses implement air conditioning
- Proper use is left to the developers ☺

Parameterization

- *Array* of 10 *integers* : `int[10]`
- *Array* of 10 *booleans* : `boolean[10]`



Parameterized classes

- *List* of *Integers* : `ArrayList<Integer>`
- *Collection* of *Books* : `HashMap<String,Book>`

Test.java – check things out

```
package isical;

import java.util.ArrayList;

public class Test {

    public static void main(String[] args) {
        ArrayList<String> arrayList = new ArrayList<String>();
        arrayList.add("First"); arrayList.add("Second");
        arrayList.add("Third"); arrayList.add("Another");
        arrayList.add("Yet another");

        System.out.println(arrayList.size());
        System.out.println(arrayList.get(3));
        System.out.println(arrayList.get(8));

        arrayList.set(4, "XYZ");
        System.out.println(arrayList.get(4));
    }
}
```

Also check out HashMap

```
import java.util.HashMap;

// REST OF THE BODY OF THE CLASS SHOULD BE HERE…

HashMap<Integer,String> map = new HashMap<Integer,String>();
        map.put(1, "Prateek Pandey");
        // Put your roll numbers

        // Now get the value given the key
        System.out.println("Roll number: " + 1 +
                                ", Name: " + map.get(1));
```

A Basic Name Directory

```
package isical;

/**
 * This class stores the names of students per batch
 */
public abstract class NameDirectory {

	/**
	 * Adds a student to the specified batch
	 *
	 * @param student
	 * @param batchName
	 */
	public abstract void addStudent(Student student, String batchName);

	/**
	 * Given a batch name and student name, finds the student in the batch
	 *
	 * @param batchName
	 * @param studentName
	 * @return the student object if found, null otherwise
	 */
	public abstract Student getStudent(String batchName, String studentName);
}
```

Implementation : DynNameDirectory

```
package isical;

public class DynNameDirectory extends NameDirectory {
```

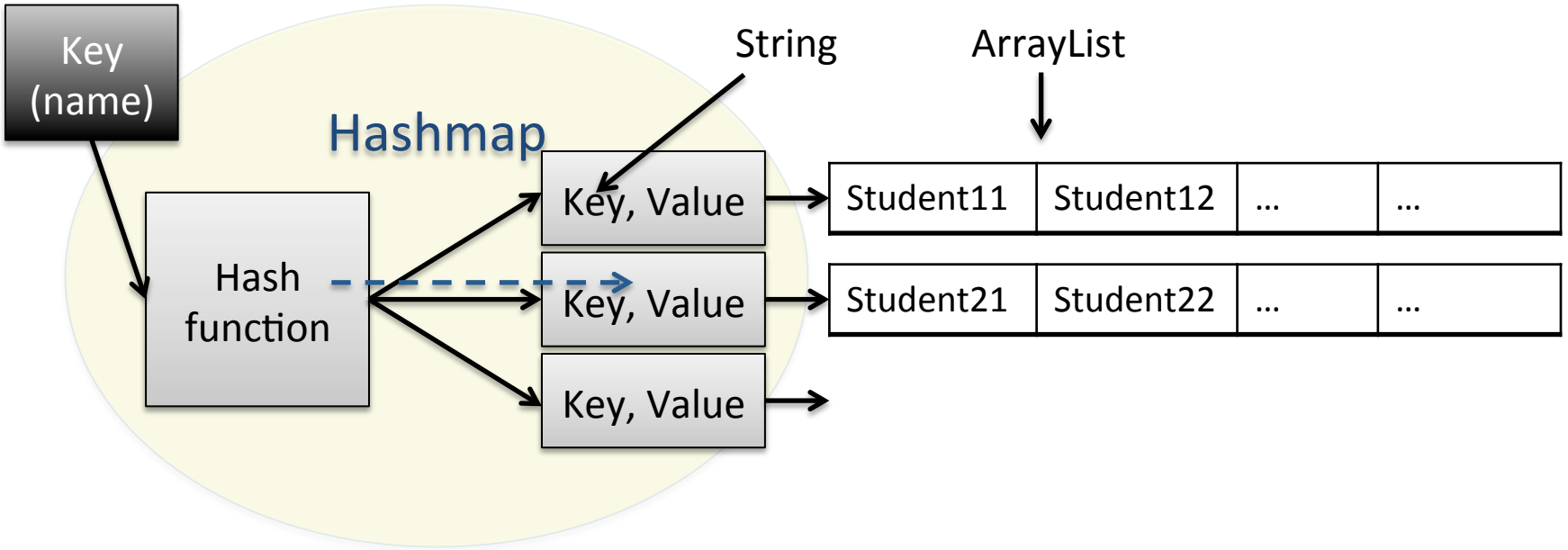

```
    @Override
    public void addStudent(Student student, String batchName) {
        // TODO: Implement it
    }

    @Override
    public Student getStudent(String batchName, String studentName) {
        // TODO: Implement it
        return null;
    }

}
```

Implementation : DynNameDirectory

```
package isical;

public class DynNameDirectory extends NameDirectory {

    // Internal storing mechanism
    private HashMap<String,ArrayList<Student>> directory
        = new HashMap<String,ArrayList<Student>>();

    @Override
    public void addStudent(Student student, String batchName) {
        // TODO: Implement it
    }

    @Override
    public Student getStudent(String batchName, String studentName) {
        // TODO: Implement it
        return null;
    }

}
```

Implementation : DynNameDirectory

```
package isical;

public class DynNameDirectory extends NameDirectory {

    // Internal storing mechanism
    private HashMap<String,ArrayList<Student>> directory
        = new HashMap<String,ArrayList<Student>>();
```

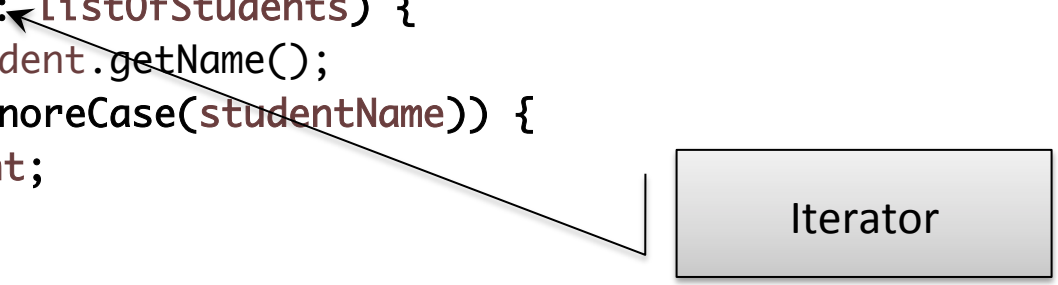


Implementation : addStudent

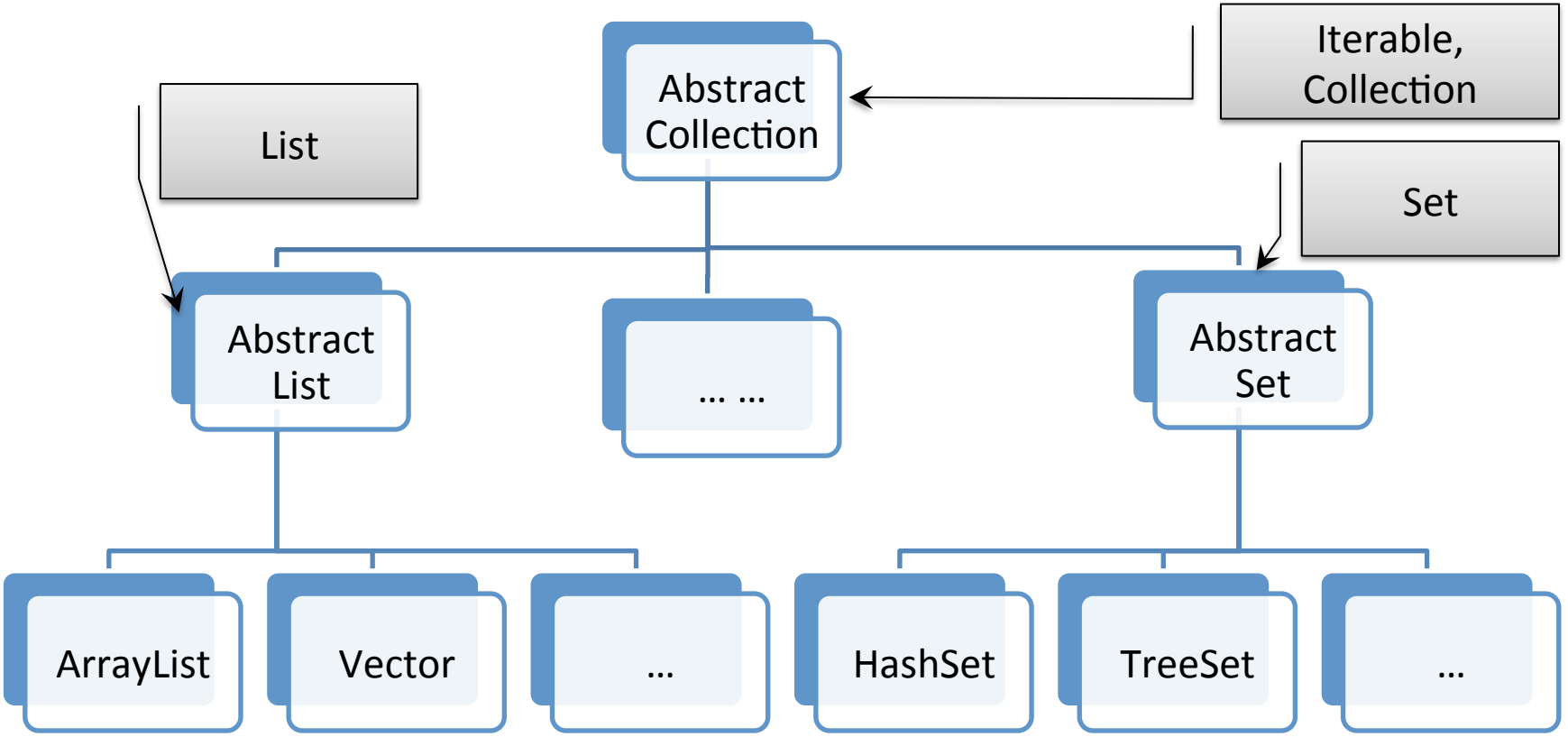
```
package isical;

public class DynNameDirectory extends NameDirectory {

    private HashMap<String,ArrayList<Student>> directory
        = new HashMap<String,ArrayList<Student>>();

    @Override
    public void addStudent(Student student, String batchName) {
        // Check if the batch exists
        ArrayList<Student> list = directory.get(batchName);
        // If does not exist, should return null
        if (list == null) {
            list = new ArrayList<Student>();
        }
        // Now add the student
        list.add(student);
        // Now put the list back into the hash map
        directory.put(batchName, list);
    }
}
```

Implementation : getStudent

```
public class DynNameDirectory extends NameDirectory {
    private HashMap<String,ArrayList<Student>> directory
        = new HashMap<String,ArrayList<Student>>();

    @Override
    public Student getStudent(String batchName, String studentName) {
        // Get the list corresponding to the batch
        ArrayList<Student> listOfStudents = directory.get(batchName);
        // If does not exist, then return null anyway
        if (listOfStudents == null) return null;
        // Otherwise find the student by iterating over the students
        for (Student student : listOfStudents) {
            String name = student.getName();
            if (name.equalsIgnoreCase(studentName)) {
                return student;
            }
        }
        // If nobody is found, return null
        return null;
    }
}
```



Collections, Lists



Generics

```
package isical;

/**
 * A pair of two elements. First one is of type T1, the second one is of type T2
 *
 * @param <T1>
 * @param <T2>
 */
public class Pair<T1,T2> {
     private T1 element1;
     private T2 element2;

     public Pair(T1 e1, T2 e2) {
          element1 = e1; element2 = e2;
     }

     public T1 getFirstElement() {
          return element1;
     }

     public T2 getSecondElement() {
          return element2;
     }
}
```

Write the methods to *set* the first and second elements

Is one pair comparable to another?

Generics

```
package isical;

public class Pair<T1,T2> {
     private T1 element1;
     private T2 element2;


}
```

What do we need for comparing?

```
public class Pair<T1 extends Comparable<T1>,T2>

public class Pair<T1 extends Comparable<T1>,T2> implements Comparable<Pair<T1,T2>> {

     @Override
     public int compareTo(Pair<T1, T2> o) {
          // Simply compare based on element1, because T1 is comparable
          return element1.compareTo(o.getFirstElement());
     }

}
```

Advanced in-class Exercise

- Implement your own List, called MyList with basic dynamic element addition feature
- Should support methods to
  - Check current size (number of elements)
  - Add an element
  - Get the element at position i
  - Set an element at position i
- Generic class definition:

```
public class MyList<T> {

}
```