

CSE 12 – Basic Data Structures and Object-Oriented Design

Lecture 23

Greg Miranda & Paul Cao, Winter 2021

Announcements

- Quiz 23 due Monday @ 8am
- Survey 9 due tonight @ 11:59pm
- PA8 due Thursday, March 11th @ 11:59pm
 - No resubmission
- PA7 Resubmission due Friday, March 12th @ 11:59pm
- Final Exam
 - Starts Saturday, March 13th @ 8:00am
 - Ends Monday, March 15th @ 11:59pm
 - 3 hour exam – clock starts when you open exam
 - Must be finished in one sitting

Topics

- Questions on Lecture 23?
- Combine Streams
- wildcard intro (review)

Combining Streams

- How could we use streams with other streams?
- live coding demo

Motivation for wildcard

```
class Person{  
    private String name;  
    public Person(){  
        name = "paul";  
    }  
    public Person(String name){  
        this.name = name;  
    }  
    public String toString(){  
        return name;  
    }  
}
```

```
class Student extends Person{  
    private int credits;  
    public Student(){  
        super() ;  
        credits = 12;  
    }  
    public Student(String name, int credits){  
        super(name) ;  
        this.credits = credits;  
    }  
    public String toString(){  
        return super.toString() + " " + credits;  
    }  
}
```

```

class Person{
    private String name;
    public Person(){
        name = "paul";
    }
    public Person(String name){
        this.name = name;
    }
    public String toString(){
        return name;
    }
}

class Student extends Person{
    private int credits;
    public Student(){
        super();
        credits = 12;
    }
    public Student(String name, int credits){
        super(name);
        this.credits = credits;
    }
    public String toString(){
        return super.toString() + " " + credits;
    }
}

```

```

public class WildCardsExe <E extends Person>{

    public E findFirst(ArrayList<E> list){
        if (list == null || list.size() == 0){
            return null;
        }
        return list.get(0);
    }

    public static void main(String[] args){
        WildCardsExe<Student> ref = new WildCardsExe<Student>();
        ArrayList<Person> pList = new ArrayList<Person>();
        pList.add(new Person("PC"));
        pList.add(new Person("GM"));
        System.out.println(ref.findFirst(pList));
        ArrayList<Student> sList = new ArrayList<Student>();
        sList.add(new Student("PC", 11));
        sList.add(new Student("GM", 33));
        System.out.println(ref.findFirst(sList));
    }
}

```

What will happen when we try to run this code?

- A. PC
PC 11
- B. GM
GM 33
- C. Compiler error
- D. Runtime error

```

class Person{
    private String name;
    public Person(){
        name = "paul";
    }
    public Person(String name){
        this.name = name;
    }
    public String toString(){
        return name;
    }
}

class Student extends Person{
    private int credits;
    public Student(){
        super();
        credits = 12;
    }
    public Student(String name, int credits){
        super(name);
        this.credits = credits;
    }
    public String toString(){
        return super.toString() + " " +
credits;
    }
}

```

```

public class WildCardsExe <E extends Person>{

    public E findFirst(ArrayList<E> list){
        if (list == null || list.size() == 0){
            return null;
        }
        return list.get(0);
    }

    public static void main(String[] args){
        WildCardsExe<Student> ref = new WildCardsExe<Student>();
        ArrayList<Person> pList = new ArrayList<Person>();
        pList.add(new Person("PC"));
        pList.add(new Person("GM"));
        System.out.println(ref.findFirst(pList));
        ArrayList<Student> sList = new ArrayList<Student>();
        sList.add(new Student("PC", 11));
        sList.add(new Student("GM", 33));
        System.out.println(ref.findFirst(sList));
    }
}

```

If I change the red line to the following, what will happen?

```
WildCardsExe<Person> ref = new WildCardsExe<Person>();
```

- A. PC
PC 11
- B. GM
GM 33
- C. Compiler error
- D. Runtime error

Wildcards

Hope

- Our generic class should take any type that is a subtype of `E`
- And we hope that `findFirst` can take `ArrayList` of any subtype of `E`

But

- Current generic system doesn't allow that.

```
public class WildCardsExe <E extends Person>{  
    public E findFirst(ArrayList<E> list)
```

Java provides a flexible type – the wildcard – ?

`<?>` means any type

`Collection<?>` means Collection of any type


```
public class WildCardsExe <E extends Person>{

    public E findFirst(ArrayList<? extends E> list){
        if (list == null || list.size() == 0){
            return null;
        }
        return list.get(0);
    }
    public static void main(String[] args){
        WildCardsExe<Person> ref = new WildCardsExe<Person>();
        ArrayList<Person> pList = new ArrayList<Person>();
        pList.add(new Person("PC"));
        pList.add(new Person("HA"));
        System.out.println(ref.findFirst(pList));
        ArrayList<Student> sList = new ArrayList<Student>();
        sList.add(new Student("PC", 11));
        sList.add(new Student("HA", 33));
        System.out.println(ref.findFirst(sList));
    }
}
```

? : unbounded wildcard represents any subtype of E so our ArrayList is more general (it implies ? extends Object)

? extends E : bounded wildcard represents E or any subtype of E

? super E: lower-bounded wildcard represents E or any super type of E

```
void doIt(Collection<? extends Student> data){  
    for (Student s: data){  
        System.out.println(s)  
    }  
}
```

Does the following code compile?

```
Collection<Student> data = new ArrayList<Student>();  
doIt(data);
```

A. Yes

B. No

Does the following code compile?

```
Collection<Person> data = new ArrayList<Person>();  
doIt(data);
```

```
void doIt(Collection<? extends Student> data){  
    for (Student s: data){  
        System.out.println(s)  
    }  
}
```

Does the following code compile?

```
Collection<Student> data = new ArrayList<Student>();  
doIt(data);
```

Does the following code compile?

```
Collection<Person> data = new ArrayList<Person>();  
doIt(data);
```

How do we change doIt such that it will work for both situations

- A. change parameter to `Collection<? extends Person> data`
- B. change parameter to `Collection<? super Student> data`
- C. change parameter to `Collection<? super Person> data`
- D. change foreach loop to `for (Object s: data)`
- E. Some combination of the above

Unbounded wildcard – ‘?’

```
static void soundOff(Collection<?> listOfAnimals) {  
  
    for (Animal a : listOfAnimals) {  
  
        a.makeNoise();  
  
    }  
  
}
```

```
Collection<Dog> dogList = new ArrayList<Dog>();  
soundOff(dogList);
```

Does this solve our problem?

- A. Yes, this code will work
- B. No, this code has a compile error

addAll should accept collections that contain any type that 'is-a' E.

```
public abstract class AbstractCollection<E>
    implements Collection<E> {
    // Add all the elements of the argument Collection
    // to this Collection
    public boolean addAll(_____ c) {
```

- A. Collection<E>
- B. Collection<?>
- C. Collection<? extends E>
- D. Collection<? super E>
- E. More than one of these will work

```
import java.util.*;

public class SuperWildCardDemo {

    public static void main(String[] args) {

        ArrayList<String> list1 = new ArrayList<String>();

        ArrayList<Object> list2 = new ArrayList<Object>();

        list2.add("CSE");

        list2.add(12);

        list1.add("UCSD");

        add(list1, list2);

        System.out.println(list2);

    }
```

What types should I fill into the blanks

- | | |
|-------------------|-------------|
| A. T | ? extends T |
| B. ? extends T | T |
| C. T | ? super T |
| D. ? super T T | |
| E. Something else | |

```
public static <T> void add(ArrayList<_____> c1, ArrayList<_____> c2) {

    while (!c1.isEmpty())

        c2.add(c1.remove(0));

}
```