INFO1113 Object-Oriented Programming

Week 11B: Wildcards

Slides developed by Tyson Thomas

Copyright Warning

Topics

- Wildcards (s. 4)
- **extends** with Wildcard (s. 16)
- **super** with Wildcard (s. 26)

Welcome back to generics!

Wildcards

We are going back to using generics but we will be exploring the wildcards a little further.

Unlike arrays, where we are able to assign types to a variable of an inherited type, Generic types cannot be assigned to a type that specifies a lower bound.

Can we do the same with generics?

Wildcards

The answer is **No**, otherwise we run into a similar issue of being able to cast to a collection variable of a parent type.

However, we are able to **read** super types using wildcards and **write** to a list knowings it lower bound.

Wildcards

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + "," + employeeID + "]"; }
}

public class GenericsWildcard {

    public static void readPeople(List<Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

Wildcards

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + "," + employeeID + "]"; }
}

public class GenericsWildcard {

    public static void readPeople(List<Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

Creating an **ArrayList** that contains Employee objects. We can see that Employee is a type of **Person**.

Wildcards

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + "," + employeeID + "]"; }
}

public class GenericsWildcard {

    public static void readPeople(List<Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

Given the List requires person the compiler will refuse to accept a list that contains a subtype.

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using ? symbol. This will allow the many different types to be associated with the container.

Syntax:

```
Type<?> variable;
Type<? super LowerBound> variable;
Type<? extends UpperBound> variable;
```

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using ? symbol. This will allow the many different types to be associated with the container.

Syntax:

```
Type<?> variable;
Type<? super LowerBound> variable;
Type<? extends UpperBound> variable;
```

Example:

```
List<?> list;
List<? extends Person> people;
List<? super Employee> employees;
```

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using ? symbol. This will allow the many different types to be associated with the container.

Syntax:

```
Type<?> variable;
Type<? super LowerBound> variable;
Type<? extends UpperBound> variabl
```

Given an upper bound, any type we retrieve from this collection can be treated as the **upper bound**.

Example:

```
List<?> list;
List<? extends Person> people;
List<? super Employee> employees;
```

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using ? symbol. This will allow the many different types to be associated with the container.

Syntax:

```
Type<?> variable;
Type<? super LowerBound> variable;
Type<? extends UpperBound> variabl
```

Example:

```
List<?> list;
List<? extends Person> people;
List<? super Employee> employees;
```

Given a collection which can contain its lower bound, we can only assume the types we read from this will be of Object but we can add **Employee** objects to it.

Wildcards

Given some class that utilises generics, we are able to specify a wildcard by using ? symbol. This will allow the many different types to be associated with the container.

Syntax:

```
Type<?> variable;
Type<? super LowerBound> variable;
Type<? extends UpperBound> variabl
```

We specified no lower or upper bound, this is a wild card which we can only read from and treat only as an object,

Example:

```
List<?> list;
List<? extends Person> people;
List<? super Employee> employees;
```

So how do we read all the employees?

Wildcards

Remembering the pattern with wildcards.

Producer extends, Consumer super

(PECS)

Given a collection which specifies an upper bound, we specifically use the collection for reading only.

Given a collection which specifies a lower bound, only writing performed on it.

Wildcards

Okay! Let's fix this

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + "," + employeeID + "]"; }
}

public class GenericsWildcard {

    public static void readPeople(List<? extends Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

Wildcards

Okay! Let's fix this

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + ","
}

public class GenericsWildcard {

    public static void readPeople(List<? extends Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

We have used a **wildcard** and provided an upper bound for the generic.

Wildcards

Okay! Let's fix this

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + ","
}

public class GenericsWildcard {

    public static void readPeople(List<? extends Person> people) {
        for(Person p : people) {
            System.out.println(p);
        }
    }

    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee("Jeff", 20, 76559));
        employees.add(new Employee("Alice", 20, 27584));
        employees.add(new Employee("Kelly", 32, 4332));
        readPeople(employees);
    }
}
```

Since the upper bound has been specified we are able to treat all object within this list as the upper bound

Wildcards

Okay! Let's fix this

```
class Person {
    String name;
    int age;
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String toString() { return "[" + name + "," + age + "]"; }
}

class Employee extends Person {
    int employeeID;
    public Employee(String name, int age, int employeeID) {
        super(name, age);
        this.employeeID = employeeID;
    }
    public String toString() { return "[" + name + "," + age + "," + employeeID + "]"; }
}

public class GenericsWildcard {

    public static void readPeople(List<? extends Person> people) {
        for(Person p : people) {
```

```
> java GenericsWildcard
[Jeff,20,76559]
[Alice,20,27584]
[Kelly,32,4332]
<Program End>
```

Why is it a problem to write an object to a covariant collection?

Wildcards

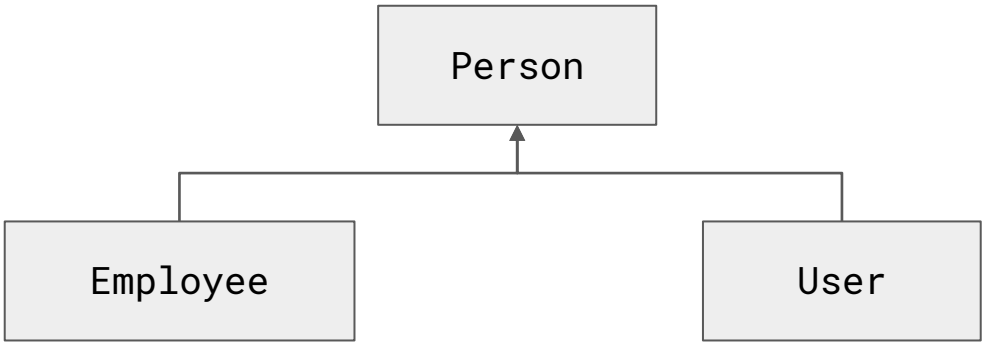
Let’s consider the following

```
List<? extends Person> people;
```

We could assign this variable to any list type that has a type argument which is a sub-type of Person.

Wildcards

But what if we have an inheritance hierarchy like so?



The following can be assigned to:

```
List<? Extends Person> people = new ArrayList<Employee>();
List<? Extends Person> people = new ArrayList<User>();
List<? Extends Person> people = new ArrayList<Person>();
```

Wildcards

However, we couldn’t add any objects to the list because we cannot assume what type the variable is bound to.

Otherwise we could add **User** to a list that contains **Employee** or **Person** objects to **User** lists.

Okay, so what about the super keyword?

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<Media> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<Media> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

Let's consider the case where we have two collection types, one contains Media and the other contains Books.

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<Media> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

We want to add Book objects to both lists

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<Media> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

However, the **addBook** method only accepts a list with the type argument of **Media**.

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}
```

However, the **addBook** method only accepts a list with the type argument of **Media**.

```
> Javac GenericWrite.java
error: incompatible types: List<Book> cannot be converted to List<Media>
        addBook(b, new Book("War and Peace", 1225));
                ^
Note: Some messages have been simplified; recompile with -Xdiags:verbose
to get full output
1 error
```

We can specify a lower bound on the list

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<? super Book> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<? super Book> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

Given the list must be able to contain a **Book** we specify parameter to accept any list which can contain Book and its super types.

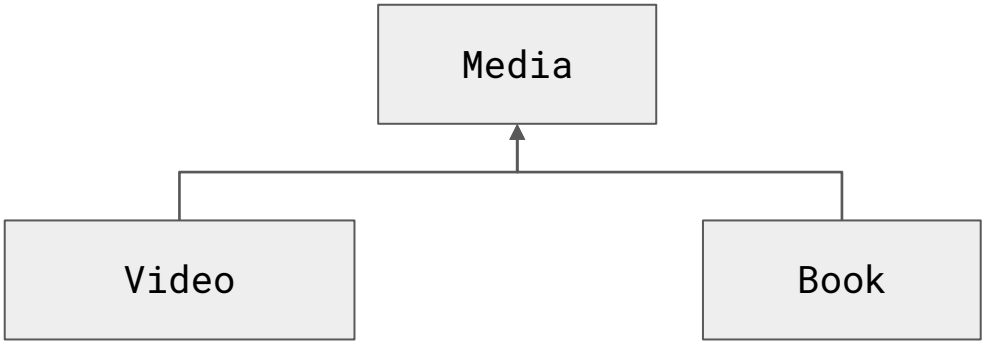
Wildcards

```
class Media {
    String title;
    public Media(String title) { this.title = title; }
}

class Book extends Media {
    int pageCount;
    public Book(String name, int pageCount) {
        super(name);
        this.pageCount = pageCount;
    }
}
class Video extends Media {
    double duration;
    public Video(String name, double duration) {
        super(name);
        this.duration = duration;
    }
}

public class GenericWrite {
    public static void addBook(List<? super Book> media, Book b) {
        media.add(b);
    }
    public static void main(String[] args) {
        List<Media> m = new ArrayList<Media>();
        List<Book> b = new ArrayList<Book>();

        addBook(m, new Book("Pride and Prejudice", 432));
        addBook(b, new Book("War and Peace", 1225));
    }
}
```

We can pass List<Media> and List<Book> to the addBook method without any compilation error.

Wildcards

Considering the inheritance heirarchy, we are able to specify the lower bound which can add the lower bound but nothing else.

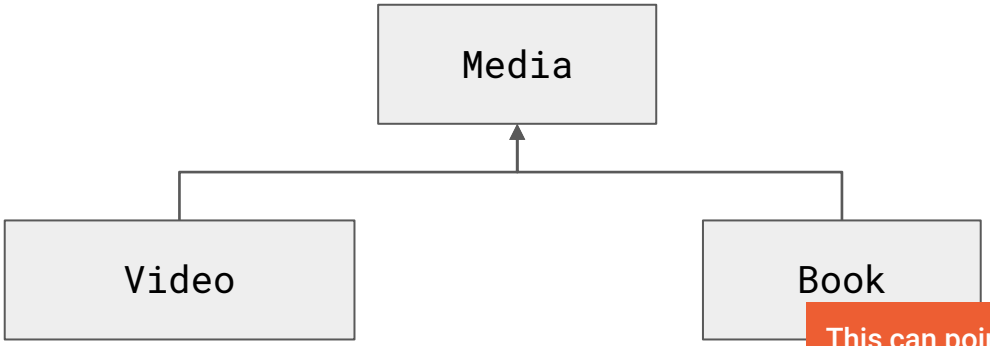


The following can be assigned to:

```
List<? super Book> books = new ArrayList<Book>();
List<? super Book> media = new ArrayList<Media>();
```

Wildcards

Considering the inheritance heirarchy, we are able to specify the lower bound which can add the lower bound but nothing else.



The following can be assigned to:

```
List<? super Book> books = new ArrayList<Book>();
List<? super Book> media = new ArrayList<Media>();
```

This can point to a Media list and therefore the list can contain Media types within it, however since it can also point to a Book list, we cannot add a type that isn't the lower bound.

See you next time!