

Wildcard and Inference

Preliminary

The following questions are not going to be discussed. Answers is given without explanation. You are encouraged to discuss them on Ed.

1. Consider the following classes.

Animals

```
import java.util.List;

interface Trainable { }

class Animal { }
class Mammal extends Animal { }
class Dog extends Mammal implements Trainable, Comparable<Mammal> {
  @Override
  public int compareTo(Mammal m) {
    return 1;
  }
}
class ShihTzu extends Dog { }

class A {
  static <U, T extends U> U foo(List<? super T> list) {
    U u = null;
    return u;
  }
}
```

Further consider the following variable declarations.

Code Snippet

```
Dog dog = new Dog();
List<Dog> dogList = List.of(dog);
```

(a) For each of the following subtyping relationship, indicate if they are `true` or `false`.

Subtyping	Answer
`Animal` <: `Trainable`	
`Dog` <: `Comparable<Animal>`	

`ShihTzu` <: `Comparable<Animal>`	
`ShihTzu` <: `Comparable<? extends Animal>`	
`ShihTzu` <: `Comparable<? super ShihTzu>`	
`Comparable<ShihTzu>` <: `Comparable<? extends Trainable>`	

(b) For each of the following assignment, indicate if they would (i) compile without warning, (ii) compile with warning, or (iii) not compile.

Assignment	**Answer**
`dog = A.<Dog, ShihTzu>foo(dogList);`	
`dog = A.<ShihTzu, ShihTzu>foo(dogList);`	
`dog = A.<ShihTzu, Dog>foo(dogList);`	
`dog = A.<Dog, Dog>foo(dogList);`	

(c) Consider the method invocation `dog = A.foo(dogList)`.

(i) What is the type constraint from target typing?
(ii) What is the type constraint from argument typing?
(iii) What is the type constraint from type parameter bound?
(iv) What is the type inferred for the following type parameter?

- `U`
- `T`

Discussion

The following questions are problem sets to be discussed during recitation. You are expected to attempt them before coming to the recitation. You may be randomly called to present your answer during recitation. You are encouraged to discuss them on Ed.

1. Recall that `Integer` <: `Number` <: `Object`. `ArrayList<T>` is a class provided by the Java library and it implements the interface `List<T>`.

   We want to draw a diagram to depict the subtype relationships between the types. If `S` $\neq$ `T` and `S` <: `T`, then draw the following.

   `T`
   |
   `S`

   Omit any subtyping relationship that comes from **transitivity**. In other words, for three distinct types `S` <: `T` <: `U`, you do not need to draw the relationship `S` <: `U`. Draw the diagram to depict the relationships between the following types.

   - `List<Integer>`
   - `List<Object>`
   - `List<Number>`
   - `List<?>`
   - `List<? super Integer>`
   - `List<? extends Integer>`
   - `List<? super Object>`
   - `List<? extends Object>`
   - `List<? super Number>`
   - `List<? extends Number>`
   - `ArrayList<Integer>`
   - `ArrayList<Object>`
   - `ArrayList<Number>`

2. The following static generic method max takes in a list of generic type `T`, where `T` implements the `Comparable<T>` interface.

Generic Method Max

```
static <T extends Comparable<T>> T max(List<T> list) {
  T max = list.get(0);
  if (list.get(1).compareTo(max) > 0) {
    return list.get(1);
  }
  return max;
}
```

Suppose we have a `Fruit` class that implements the `Comparable<Fruit>` interface, and an `Apple` class that is a subclass of `Fruit`.

Fruit

```
class Fruit implements Comparable<Fruit> {
  public int compareTo(Fruit f) {
    return 0; // stub
  }
}
```

Apple

```
class Apple extends Fruit { }
```

We also have two variables of type lists, `apples` of type `List<Apple>` and `fruits` of type `List<Fruit>`.

Code Snippet

```
List<Fruit> fruits = List.of(new Fruit(), new Apple());
List<Apple> apples = List.of(new Apple(), new Apple());
```

(a) For each of the following find the type inferred[1] for `T`. If the code does not compile, explain the cause of the compilation error.

(i) `Fruit f = max(fruits);`

(ii) `Fruit f = max(apples);`

(iii) `Apple a = max(apples);`

(iv) `Apple a = max(fruits);`

(b) Reimplement `max` to use the most flexible type for `T`. **Hint:** Do you need wildcards[2]?

(c) In your solution for Part (b) above, show what `T` would be inferred as in each of Subpart (i) to (iv) in Part (a) above. If the code does not compile, explain the cause of the compilation error.

Exercise

The following questions are to be discussed on Ed as no answer will be given. ***Use only concepts that you have learnt****. You are expected to attempt them before the next lecture as the concepts may be used and/or mentioned in the lecture. The concept may also be tested for assessment.*

1. The **Expression Problem** was first noticed by John C. Reynolds[3] and was brought to prominent in an email by Philip Wadler[4]. We will illustrate a simplified version of this problem in this question and the next.

(a) Create an abstract class `Shape` with a single method `area` that returns the area of the shape as a `double` with the following signature. `public abstract double area();`

(b) Create 3 concrete classes with the following specification.

(i) `Square`. The class has a single field `length`. The area is `length * length`.

(ii) `Circle`. The class has a single field `radius`. The area is `3.14 * radius * radius`.

(iii) `Rectangle`. The class has 2 fields `width` and `height`. The area is `width * height`.

(c) **(Adding Classes)** Implement the class `Triangle` which is a subclass of `Shape` and represent a right-angled triangle. The class has 2 fields `base` and `height`. The area is `base * height / 2`.

How easy/difficult is it to do so? In particular, how many other files do you need to compile/recompile?

(d) **(Adding Methods)** Implement the method `perimeter` that works for all subclasses of `Shape` to find the perimeters[5] of the given shape.

How easy/difficult is it to do so? In particular, how many other files do you need to compile/recompile?

Bibliography

[1] Oracle, "Type Inference." [Online]. Available: https://docs.oracle.com/javase/tutorial/java/generics/genTypeInference.html

[2] Oracle, "Wildcards." [Online]. Available: https://docs.oracle.com/javase/tutorial/java/generics/wildcards.html

[3] J. C. Reynolds, "User-Defined Types and Procedural Data Structures as Complementary Approaches to Data Abstraction," in *Programming Methodology: A Collection of Articles by Members of IFIP WG2.3*, New York, NY: Springer New York, 1978, pp. 309–317.

[4] P. Wadler, "The Expression Problem." [Online]. Available: https://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt

[5] Libretexts, "Perimeter and Area." [Online]. Available: https://math.libretexts.org/Courses/Highline/_College/Math/_081/_091/%3A/_CAM/_Aligned/_Textbook/01/%3A/_Perimeter/_Area/_and/_Unit/_Conversions/1.01/%3A/_Perimeter/_and/_Area

[6] M. Fowler, "Anemic Domain Model." [Online]. Available: https://www.martinfowler.com/bliki/AnemicDomainModel.html

[7] R. Martin, "Clean Architecture: A Craftsman's Guide to Software Structure and Design," in Robert C. Martin Series., Pearson Education, 2017.