CS2030S Recitation

Week 4: Problem Set 2

brian

2026-02-19

National University of Singapore

Recap

Recap: Type safety

- What is type safety?
  - Will not have type errors
  - Cannot use values in an invalid or unintended way according to their type
- How does the compiler achieve type safety?
  - Use compile time type to determine what methods are available
  - Only allow assigning subtype to supertype (methods are guaranteed to be there)

Q1a

Consider the following subtyping relationship

`subR <: R <: SuperR`

`subE <: E <: SuperE <: Exception` and a class `A`

```
1 class A {
2   R foo() throws E { ... }
3 }
```

Let `B <: A` and overrides `foo()`

Q1a

```
1 void bar(A a) {
2   try {
3     R r = a.foo();
4     // use r
5   } catch (E e) {
6     // handle exception
7   }
8 }
```

Which implementations of `foo` in `B` violate substitutability of A with B

(a) `SubR foo() throws E`

(b) `SuperR foo() throws E`

(c) `R foo() throws SubE`

(d) `R foo() throws SuperE`

Q1a

(a) Is ok. `SubR <: R` so it can bind to `R`

(b) Not ok. `SuperR <: R`

- Let's say we allow the bind to happen. What if we do `r.g()` where `g` is found in `R` but not in `SuperR`

(c) Is ok. `SubE <: E` so it can bind to `E`

- Catching exception is like binding the exception thrown to the one declared in the catch block

(d) Not ok. `SuperE <: E` (Similar reason to (b))

Q2 background

Java provides an abstract class called `Number`

This is the superclass of all primitive *numeric* wrapper classes

`BigInteger` is a class which supports arbritrary-precision integers (giant numbers) `BigInteger` implements the `Comparable<BigInteger>` interface
Therefore,

- `BigInteger <: Number`
- `BigInteger <: Comparable<BigInteger>`

Q2 background

My best friend Ah Beng wrote a method to convert an array of `BigInteger` to an array of primitive `short` values

```
1 public static short[] toShortArray(BigInteger[] a, BigInteger threshold) {
2   short[] out = new short[a.length];
3   for (int i = 0; i < a.length; i += 1) {
4     if (a[i].compareTo(threshold) <= 0) {
5       out[i] = a[i].shortValue();
6     }
7   }
8   return out;
9 }
```

He realised he needed to do the same method for `Integer` and `Double`

Q2 background

So Ah Beng wrote the following code

```
1  public static short[] toShortArray(Integer[] a, Integer threshold) {
2    short[] out = new short[a.length];
3    for (int i = 0; i < a.length; i += 1) {
4      if (a[i].compareTo(threshold) <= 0) {
5        out[i] = a[i].shortValue();
6      }
7    }
8    return out;
9  }
10 public static short[] toShortArray(Double[] a, Double threshold) {
11   short[] out = new short[a.length];
12   for (int i = 0; i < a.length; i += 1) {
13     if (a[i].compareTo(threshold) <= 0) {
14       out[i] = a[i].shortValue();
15     }
16   }
17   return out;
18 }
```

Q2 background

Ah Beng scored A+ for CS1101S he realised he's repeating code. So he wanted to generalize the methods he has written

Q2a

This was his first attempt. What's wrong? What kind of error do we get?

```
1 public static short[] toShortArray(Object[] a, Object threshold) {
2   short[] out = new short[a.length];
3   for (int i = 0; i < a.length; i += 1) {
4     if (a[i].compareTo(threshold) <= 0) {
5       out[i] = a[i].shortValue();
6     }
7   }
8   return out;
9 }
```

Q2a

- Compile error
  - `a` has compile time type of `Object[]`
  - `a[i]` has compile time type of `Object`
  - `Object` does not have `compareTo` or `shortValue` method

Q2a

Being a persistent student, Ah Beng tried another approach.

```
1 public static short[] toShortArray(Number[] a, Number threshold) {
2   short[] out = new short[a.length];
3   for (int i = 0; i < a.length; i += 1) {
4     if (a[i].compareTo(threshold) <= 0) {
5       out[i] = a[i].shortValue();
6     }
7   }
8   return out;
9 }
```

Does it work now?

Q2a

- Still compile error
  - Even though now we have access to `shortValue`
  - We don't have access to `compareTo`

Ah Beng tries again

```
1 public static short[] toShortArray(Comparable[] a, Comparable threshold) {
2   short[] out = new short[a.length];
3   for (int i = 0; i < a.length; i += 1) {
4     if (a[i].compareTo(threshold) <= 0) {
5       out[i] = a[i].shortValue();
6     }
7   }
8   return out;
9 }
```

Why is Ah Beng wrong and going to fail CS2030S?

Q2a

- Still compile error
  - We gained access to `compareTo`
  - but lost access to `shortValue`
  - We neeed access to both, is all hope lost for my bestie?

Q2b

Ah Beng discovered that Java supports generics

A type parameter can have multiple bounds using the `&` symbol

`<T extends S1 & S2>` (Only the first thing can be a class rest must be interfaces)

Help Ah Beng rewrite the method with generics

Q2b

```
1  public static <T extends Number & Comparable<T>>
2      short[] toShortArray(T[] a, T threshold) {
3    short[] out = new short[a.length];
4    for (int i = 0; i < a.length; i++) {
5      if (a[i].compareTo(threshold) <= 0) {
6        out[i] = a[i].shortValue();
7      }
8    }
9    return out;
10 }
```

- What happens after type erasure?
  - Erase to first bound
  - Cast to other bounds when need to access the method (done by compiler)

Polymorphism after type erasure

```
1 class A<T> {
2   public void fun(T x) {
3     System.out.println("A");
4   }
5 }
```

(i)

```
1 class B extends A<String> {
2   public void fun(String i) {
3     System.out.println("B");
4   }
5 }
```

Does the code compile?

Q3

- Yes it compiles.
  - Type variable `T` is being instantiated to `String`
  - In `A`, `T` is erased to `Object`
  - In `B`, `fun(Object)` is added as a bridge method (overrides `A`) that calls `fun(String)`
  - Uses **overriding** to achieve **overloading**

```
1 A<String> a = new B();
2 a.fun("2");
```

"B" would be printed

Polymorphism after type erasure

```
1 class A<T> {
2   public void fun(T x) {
3     System.out.println("A");
4   }
5 }
```

(ii)

```
1 class B extends A<String> {
2   public void fun(Object i) {
3     System.out.println("B");
4   }
5 }
```

Does the code compile?

Q3

- No it now does not compile.
  - The reason being is that `T` is instantiated to `String`
  - In `A`, `fun(T)` is erased to `fun(Object)`
  - In `B`, we also have a `fun(Object)`
  - Since we have 2 `fun(Object)` then there is a compile error

Polymorphism after type erasure

```
1 class A<T> {
2   public void fun(T x) {
3     System.out.println("A");
4   }
5 }
```

(iii)

```
1 class B extends A<String> {
2   public void fun(Integer i) {
3     System.out.println("B");
4   }
5 }
```

Does the code compile?

Q3

- Yes it would compile
  - `fun(Integer)` overloads `fun(Object)` and is the programmers intent
  - Bridge method from `A` would still be `fun(Object)`

```
1 A<String> a = new B();
2 a.fun("2");
```

"A" would be printed

The End

bye!