Sp18 CS 61B Discussion 4

Welcome!

Wayne Li

wli2@berkeley.edu

https://wayne-li2.github.io/

Announcements

- Project 1B released, due Friday 2/9
- Midterm 2/12, 8-10 PM
  - Check Piazza for room assignments
  - Material up to 2/7
- Guerilla Section this Sunday 2/11

Quiz Instructions

- If you haven’t yet, please also **neatly** put your email address **outside the name box** if you want to be emailed!
- Bubble number **41**.

Aside

WWJP?

Live Demo - WWJP?

- Let’s hope this works...

WWJP?

- 10
- Exception in thread "main" java.lang.ClassCastException: java.base/[Ljava.lang.Object; cannot be cast to java.base/[Ljava.lang.Integer;

Why?

- Type Erasure
- Compiler replaces “T” with “Object”
  - Some metadata tells compiler “This is a generic!”
- Thus, **array = (T[ ]) new Object[10];**
- Is now **array = (Object[ ]) new Object[10];**

Type Erasure cont.

- Compiler verifies generic code at compile time, but on runtime, they are just objects! Compiler generates whatever casts are necessary, whenever needed.
- Thus, the type has been “erased”.

First Call

- **arrayOfMagic.getLength()** accesses the member from inside the generic class itself.
  - Remember that the compiler erased all “T” to “Object”. No problem!

Second Call

- **arrayOfMagic.array.length** directly accesses the array field from the outside. The compiler adds a cast:
  - **((Integer[ ]) arrayOfMagic.array).length**
- This cast is **legal** at compile time (failing on runtime).
  - WHY??????

WHY??

- Buried deep in the Java Documentation, it says:
  - If S and T are both reference types, S[ ] extends T[ ] **iff** S extends T.

Moral of the Story

- **Don’t let your interns write production code.**
  - Also take CS164 to learn more about compilers!

References

- Source 1
- Source 2
- Source 3

Exam FAQ

FAQ

- Will the questions be tricky/mean?
  - No! We aim for questions that test you on your understanding of material.
- What should I put on my cheat-sheet?
  - Any rule-based things you have learned.

FAQ

- How should I study?
  - Lots of practice midterms, and make sure you fully understand the homeworks, projects, and labs.
- What if I’m behind?
  - Start now and don’t be afraid to ask for help!

Onto Discussion

The Method Selection Algorithm

Goal: determine what code runs when we write `x.methodName(y)`.

At compile-time:

1. Look at the compile-time type of x.
2. Find all methods that accept the compile-time type of y.
3. Choose the most specific method and remember its signature.

At runtime: starting from the runtime type of x, select the lowest method in the inheritance hierarchy that has the exact signature chosen at compile-time in step 3.

Example

Puppy extends Dog, Dog extends Animal, Animal extends Object.

Puppy, Dog, and Animal each have the following methods:

- public void barkAt(Animal a)
- public void barkAt(Dog d)
- public void barkAt(Puppy p)

We execute the following code:

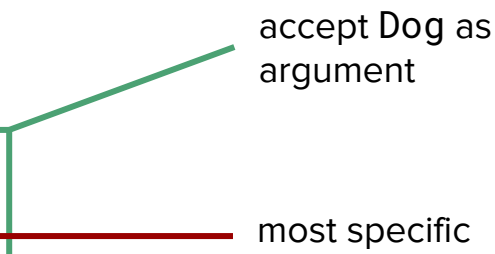
```
Dog d = new Puppy();
d.barkAt(d); // which method gets called?
```

Example (Compile-Time)

```
Dog d = new Puppy();
d.barkAt(d);
```

The compile-time type of d is Dog, so let's look at it:

```
public class Dog extends Animal {
    public void barkAt(Animal a) { … }
    public void barkAt(Dog d) { … }
    public void barkAt(Puppy p) { … }
}
```



Which methods accept a Dog as the argument?

Out of these, which is the most specific?

Example (Runtime)

```
Dog d = new Puppy();
d.barkAt(d);
```

The signature we chose at compile-time was:

```
public void barkAt(Dog d)
```

The runtime type of d is Puppy, so let's look at it:

```
public class Puppy extends Dog {
    public void barkAt(Animal a) { … }
    public void barkAt(Dog d) { … }
    public void barkAt(Puppy p) { … }
}
```

yep, found it! this is our answer

Do we see a method with the signature we chose at compile-time?