Classless Java: Interface-based Programming for the Masses

Yanlin Wang , Haoyuan Zhang , Marco Servetto , Bruno C. d. S. Oliveira

Motivation

1. Generally speaking, multiple inheritance with fields is hard. Moreover, there are problems with field type refinement, constructors, etc.
2. Java support multiple inheritance with default methods in a limited way.
3. Other models (traits, mixins, etc) have certain limitatations.
4. Our approach is related with traits. But we use operations on state to simulate fields which traits do not support.

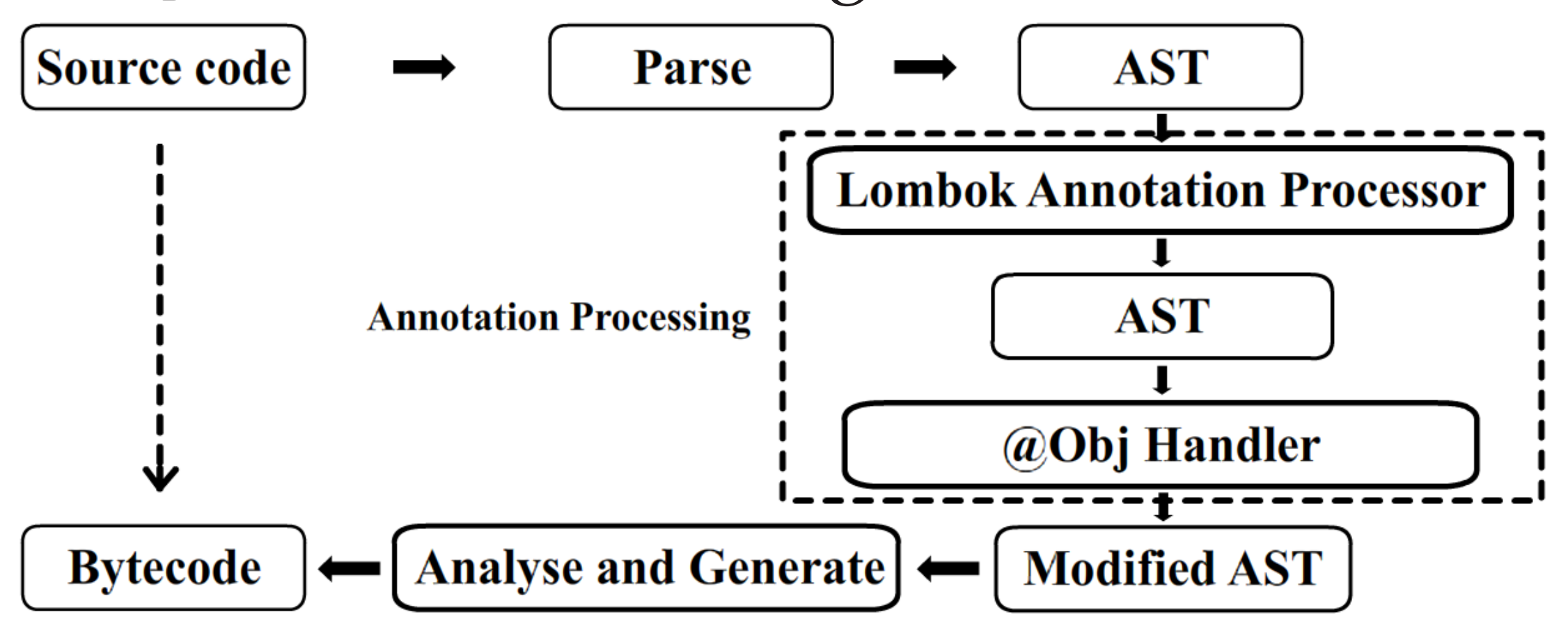
Contributions

1. IB and Object Interfaces: enable powerful programming idioms using multiple-inheritance, type-refinement and abstract state operations.
2. Classless Java: a practical realization of IB in Java.
3. Type-safe covariant mutable state
4. Applications and case studies

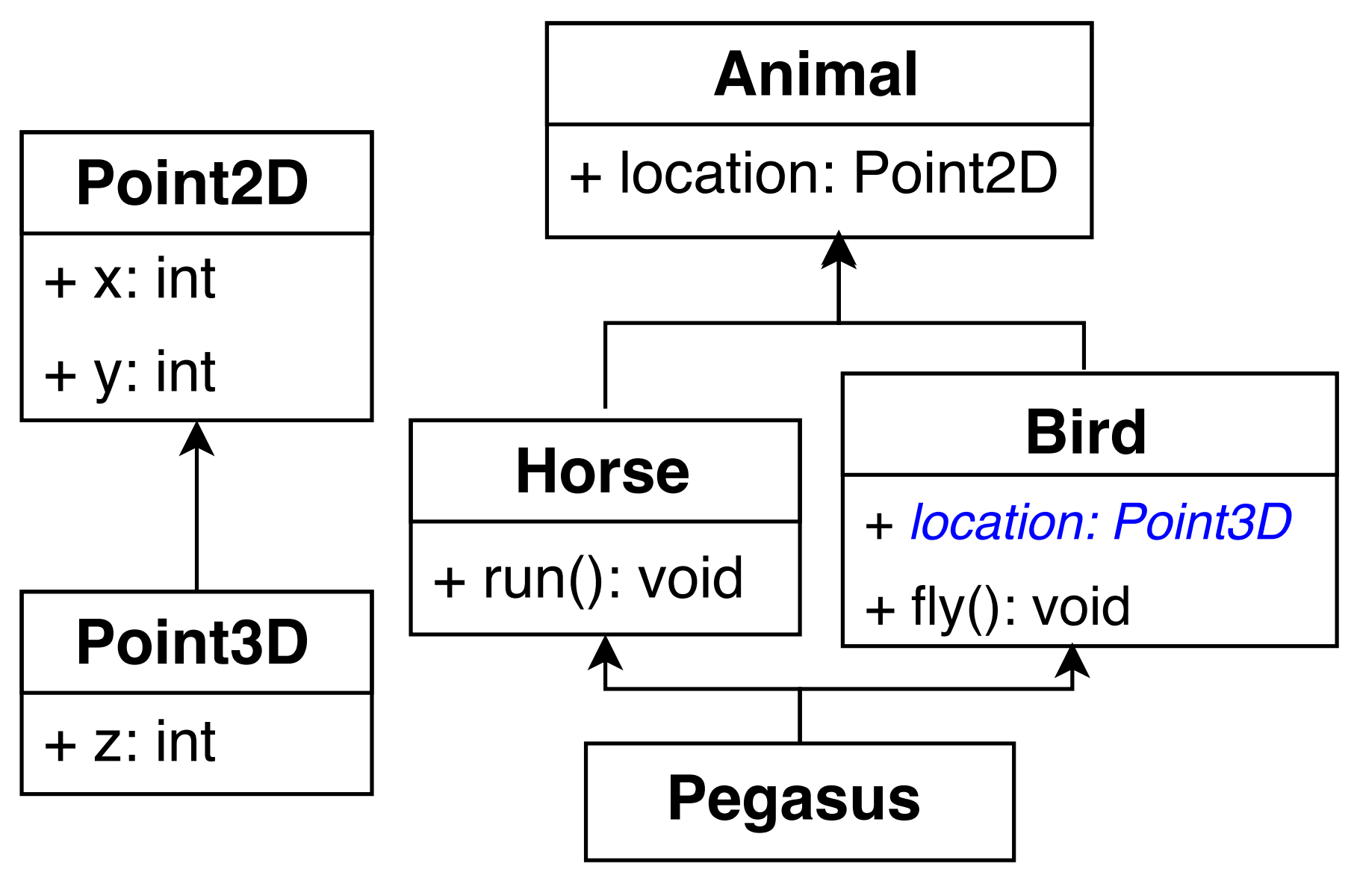
Implementation

Java supports compilation agents, where Java libraries can interact with the Java compilation process, acting as a man in the middle between the generation of AST and bytecode.

This process is facilitated by frameworks like Lombok: a Java library that aims at reducing Java boilerplate code via annotations. `@Obj` was created using Lombok.



UML Diagram



Results (Partial)

In the result of the maze game case study, both SLOC and number of interfaces are greatly reduced:

	SLOC	# of classes/interfaces
Bono et al.	335	14
Ours	199	11
Reduced by	40.6%	21.4%

Object Interfaces and Instantiation

```java
interface Animal {} // no points yet!
interface Horse extends Animal {
   default void run(){out.println("run!");}
}
interface Bird extends Animal {
   default void fly(){out.println("fly!");}
}
interface Pegasus extends Horse, Bird {}
```

```java
class HorseImpl implements Horse {}
class BirdImpl implements Bird {}
class PegasusImpl implements Pegasus {}
```

```java
@Obj interface Horse extends Animal {
  default void run() {out.println("running!");} }
@Obj interface Bird extends Animal {
  default void fly() {out.println("flying!");} }
@Obj interface Pegasus extends Horse, Bird {}
```

```java
Pegasus p = Pegasus.of();
interface Pegasus extends Horse, Bird {
  // generated code not visible to users
  static Pegasus of() { return new Pegasus() {}; }
}
```

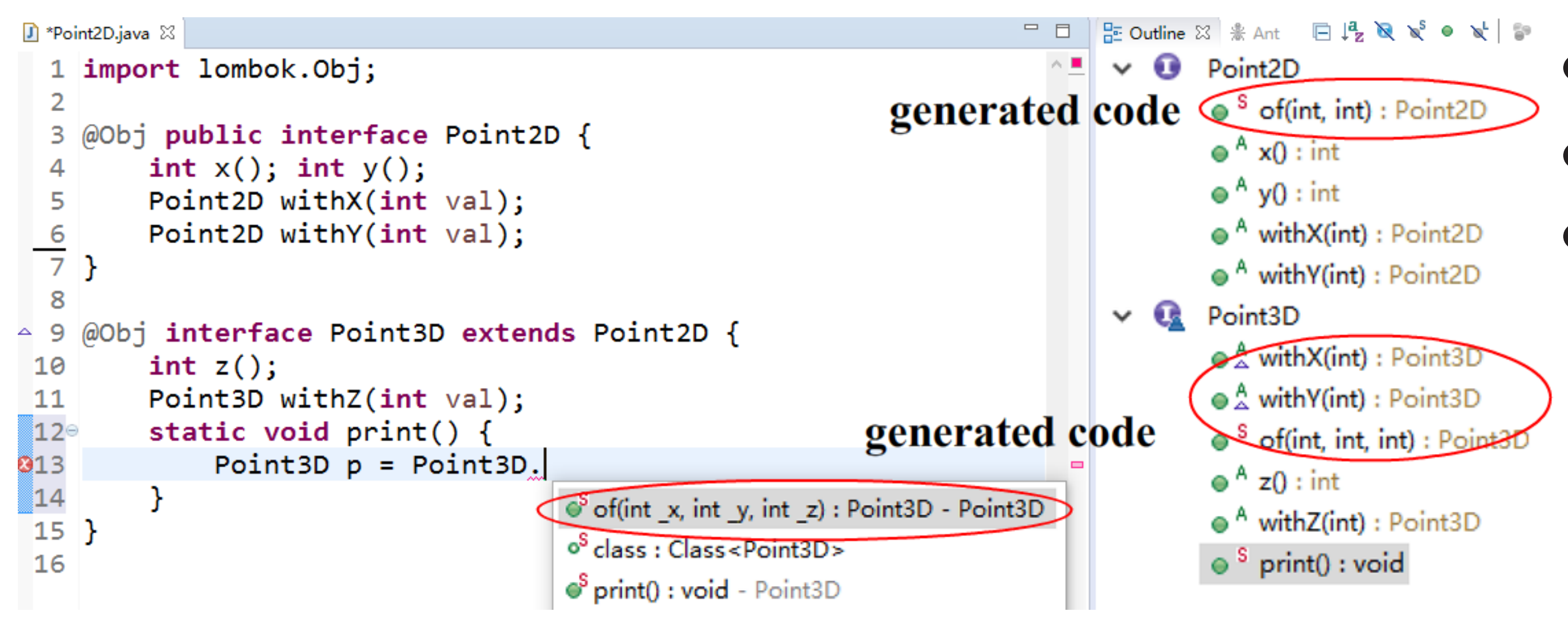
Object Interfaces with State (Immutable Data)

```java
interface Point2D { int x(); int y(); }
Point2D p = new Point2D() {
   public int x() {return 4;}
   public int y() {return 2;}
}
```

```java
@Obj interface Point2D { int x(); int y(); }
Point2D myPoint = Point2D.of(4, 2);
Point2D p = Point2D.of(42, myPoint.y());
```

More in the Paper

	Operation	Example	Description
State operations (for a field x)	**"fields"/getters**	`int x()`	Retrieves value from field x.
	withers	`Point2D withX(int val)`	Clones object; updates field x to `val`.
	setters	`void x(int val)`	Sets the field x to a new value `val`.
	fluent setters	`Point2D x(int val)`	Sets the field x to `val` and returns **this**.
Other operations	**factory methods**	`static Point2D of(int _x,int _y)`	Factory method (generated).
	functional updaters	`Point3D with(Point2D val)`	Updates all matching fields in `val`.



```java
import lombok.Obj;

@Obj public interface Point2D {
    int x(); int y();
    Point2D withX(int val);
    Point2D withY(int val);
}

@Obj interface Point3D extends Point2D {
    int z();
    Point3D withZ(int val);
    static void print() {
        Point3D p = Point3D.
    }
}
```

- Techniques for field type refinement.
- Formalization and proofs.
- Case studies and applications.
  - The Expression Problem
  - Embedded DSLs with Fluent Interfaces
  - A Maze Game
  - Refactoring an Interpreter

Results(Partial)

In the second case study, we found that embedding DSLs with our implementation of fluent interfaces is quite straightforward:

```java
ExtendedDatabase query2 =
    ExtendedDatabase.of()
    .select("a, b").from("Table")
    .where("c > 10").orderBy("b");
```

Limitations

- Current implementation only realizes ejc version.
- Generics is not fully supported.
- Separate compilation is an experimental feature with current Lombok.

Acknowledgments

This work is sponsored by the Hong Kong Research Grant Council Early Career Scheme project number 27200514. The first and second authors are supported by the SIG PAC stduent travel grant.

Contacts

The University of Hong Kong
Yanlin Wang, Haoyuan Zhang, Bruno C. d. S. Oliveira: *{ylwang,hyzhang,bruno}@cs.hku.hk*
Victoria University of Wellington
Marco Servetto: *marco.servetto@ecs.vuw.ac.nz*