Announcements/Follow-ups

- P6 due + P7 posted today
- Quiz5 tomorrow
  - Study answers posted after class
- Follow-ups
  - For-each with arrays
  - MyBigInt example
  - Cop-outs:
    - Why aren't constructors inherited?
    - Why don't generics and new mix?

Cop outs

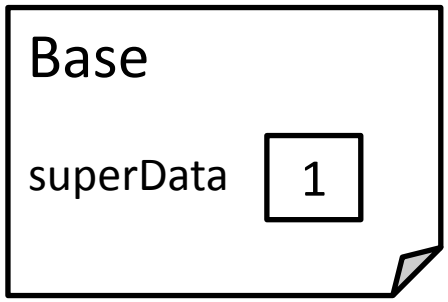
- Why aren't constructors inherited?
  - The name of the sub-constructor must be the same name as the subclass
  - The name of the super-constructor must be the same name as the subclass
  - Inherited methods aren't "copied" to the sub-class, just callable from it
- Why don't generics and `new` mix?
  - Type erasure: Type information about type parameters is not available at run-time
  - Arrays are covariant (more after `instanceof`)

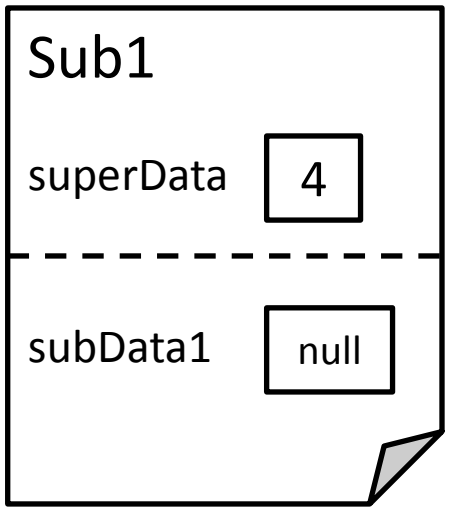
Non-cop out (via a cop out)

- A campaign promise of Java is “backwards-compatibility”
  - new releases (e.g. post generics) must work seamlessly with old releases (e.g. pre generics)
- “Non cop-out” answer: Try write a backwards-compatible Java byte-code compiler, and encounter the irresolvable issue when trying to instantiate generic arrays

Inheritance Refresher

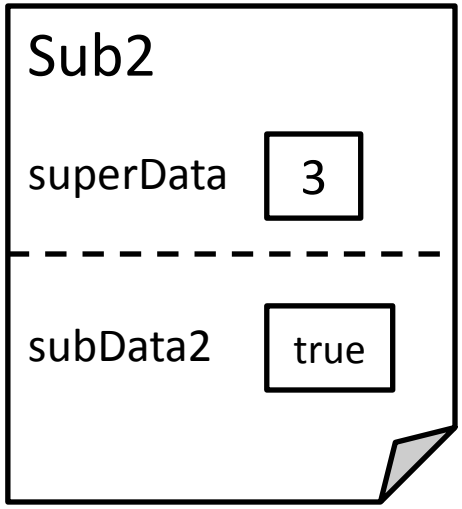
An instance of the base class





An instance of a one sub class

An instance of a different sub class



Basic syntax and usage

- The final keyword (for classes):
  - Prevents extension by any other class
  - Truly immutable classes (String, Integer, etc.) are final
    - Otherwise MyMutableString could extend String, and violate the “is-a” relationship
  - Final children are leaves in the tree of inheritance
- Casting rules: based on the “is-a” relationship again
  - Casting example
  - ClassCastException – checked or unchecked?

Casting Terminology

- For primitives:
  - Widening conversion: double x = 3;
    - “Wider” because double has more bytes than int
  - Narrowing conversion: int x = (int) 3.0;
- For references:
  - Upcast: sub-class to super-class (or interface)
  - Downcast: super-class to sub-class

Protected access level

- There is one more access modifier: `protected`.
  - Accessible from this class, other classes in the same package, and sub-classes (in any package).
  - Mommy/Daddy/Baby example

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Overriding instance methods

- Instance methods with identical declarations in a sub-class and super-class
  - sub-class version “overrides” super-class version
  - `String toString()`
  - `boolean equals(Object other)`
  - `MovingShape – override/overload redirect()`
- The `final` keyword (for methods): If the class is not final, sub-classes may exist, but they cannot override final methods.
- The analogy to static methods is called “hiding”.
- The analogy to fields is called “shadowing”.

The super keyword

- `super` is like `this`, but it refers to inherited members.
- `super` and `this` can also be used in one constructor to call another constructor
  - `this(…)` calls another constructor in the sub-class (with a different signature)
  - `super(…)` calls a constructor in the super-class (potentially with the same signature).
    - Must be the first line of constructor body
    - If omitted, `super()` without arguments is called by default. In this case, a zero-parameter constructor must be defined in the base-class, or there is a compile-time error

Super rules

- Super-class has zero-argument constructor
  - Super() can be omitted and is assumed as first line of sub-class constructor
  - Super-class with no constructors is included in this case (has default zero-argument constructor)
- Super-class has explicit constructors but no zero-argument constructor
  - Sub-class must declare an explicit constructor
  - Super(…) must be the first line, with same parameters as a super-class constructor
- reusable.MovingShape example

The `abstract` keyword

- A middle-ground between inheritance and interfaces. Useful in the following situation:
  - Some members will be identical for all sub-classes, so they should be implemented in the base class.
  - Other members must exist, but their implementations will depend entirely on the sub-classes. So they cannot be implemented in the base class.
- The `abstract` keyword identifies the second category.
- A class with abstract members must be declared abstract itself.
- AbstractShape example

The `instanceof` operator

- Similar concept to getClass(), but with important differences
- Usage: `ref1 instanceof MyClass`
  - True when the run-time type of ref1 is MyClass *or any sub-class of* MyClass
- Casting example
  - Early/Late Binding revisited
  - Dangers of non-standard equals

Inheritance and generics

- Extends can be used with type Parameters
  - LineUp<A extends Athlete>
    - Now A can't be any type, only Athlete and its sub-types
  - MyGeneric<T **extends** MyInterface>
    - "Extends" was chosen as the keyword in type-parameterization, even for interfaces
    - When your class implements an interface, you write "implements"
    - But when your interface is derives from a super-interface, you write "extends"
    - ShapeList example
- Arrays are covariant, collections are not
  - ArraysAndGenerics example

Multiple Inheritance

```
class Square
        extends
        Rhombus,
        Rectangle
{…}
```

- Issues?
- Supported in Java for interfaces, but not classes
- Supported for classes in other languages, e.g. C++