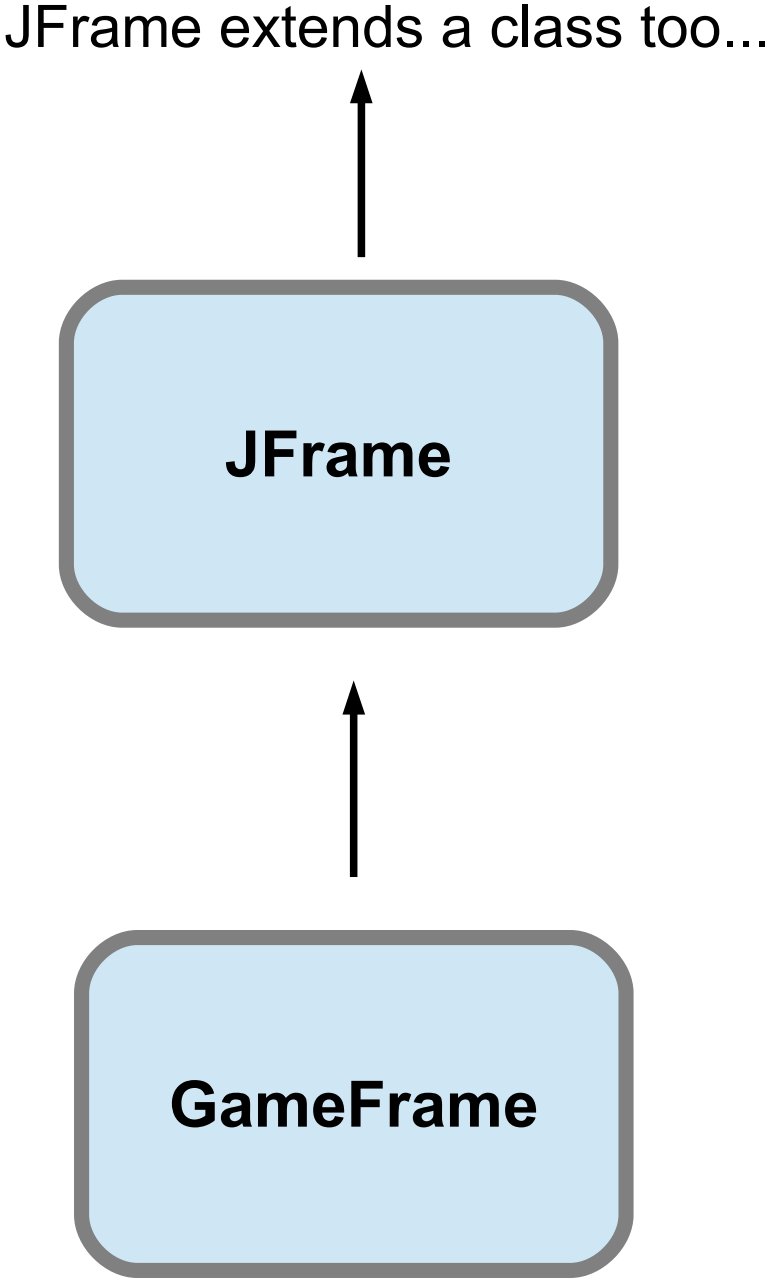
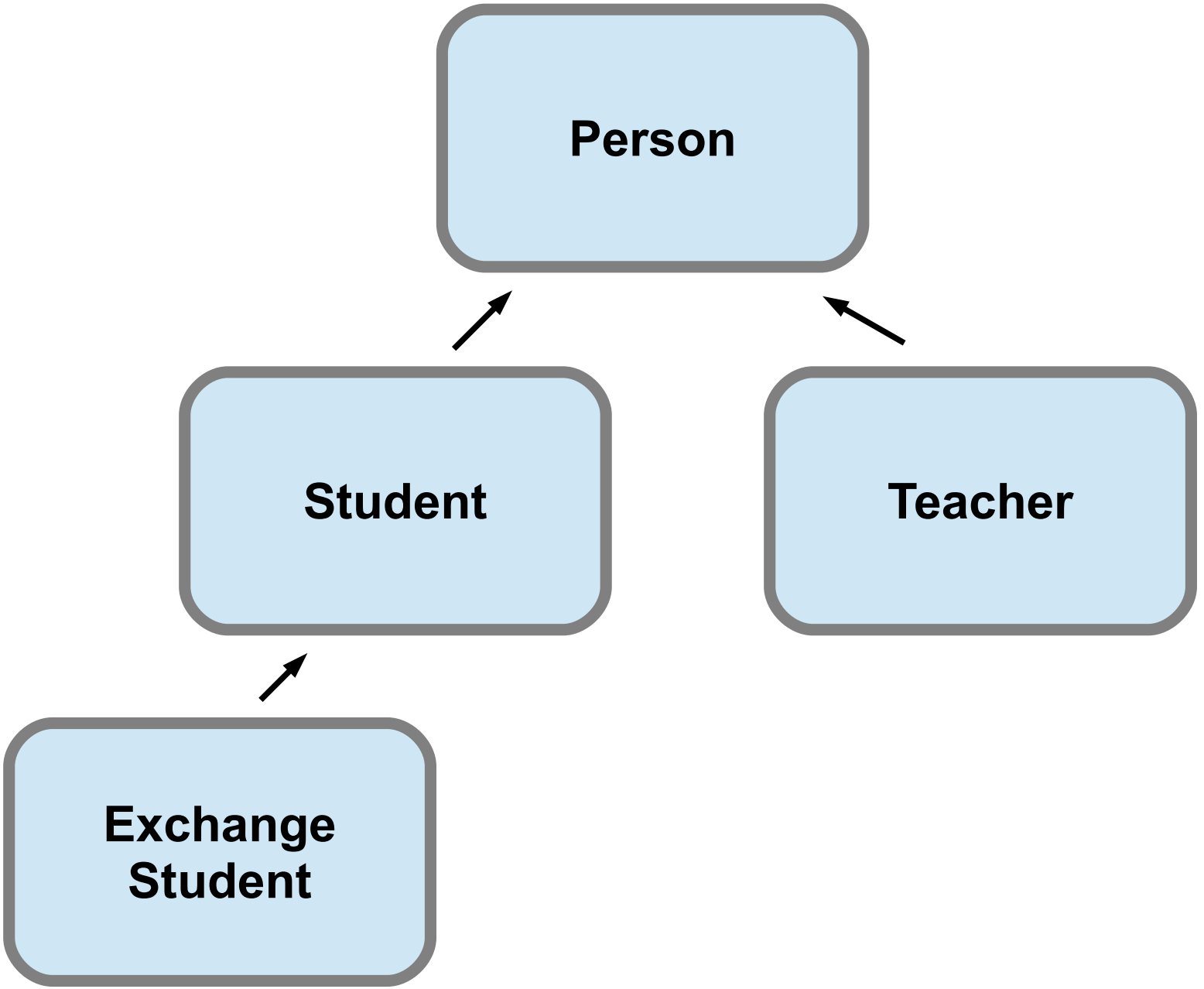
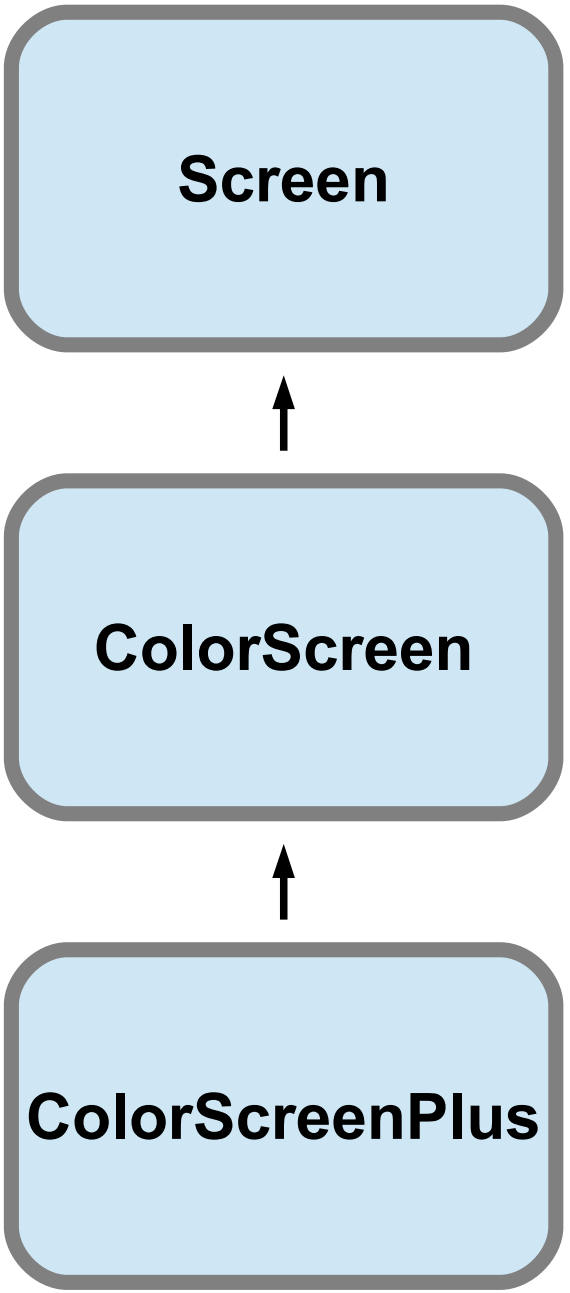
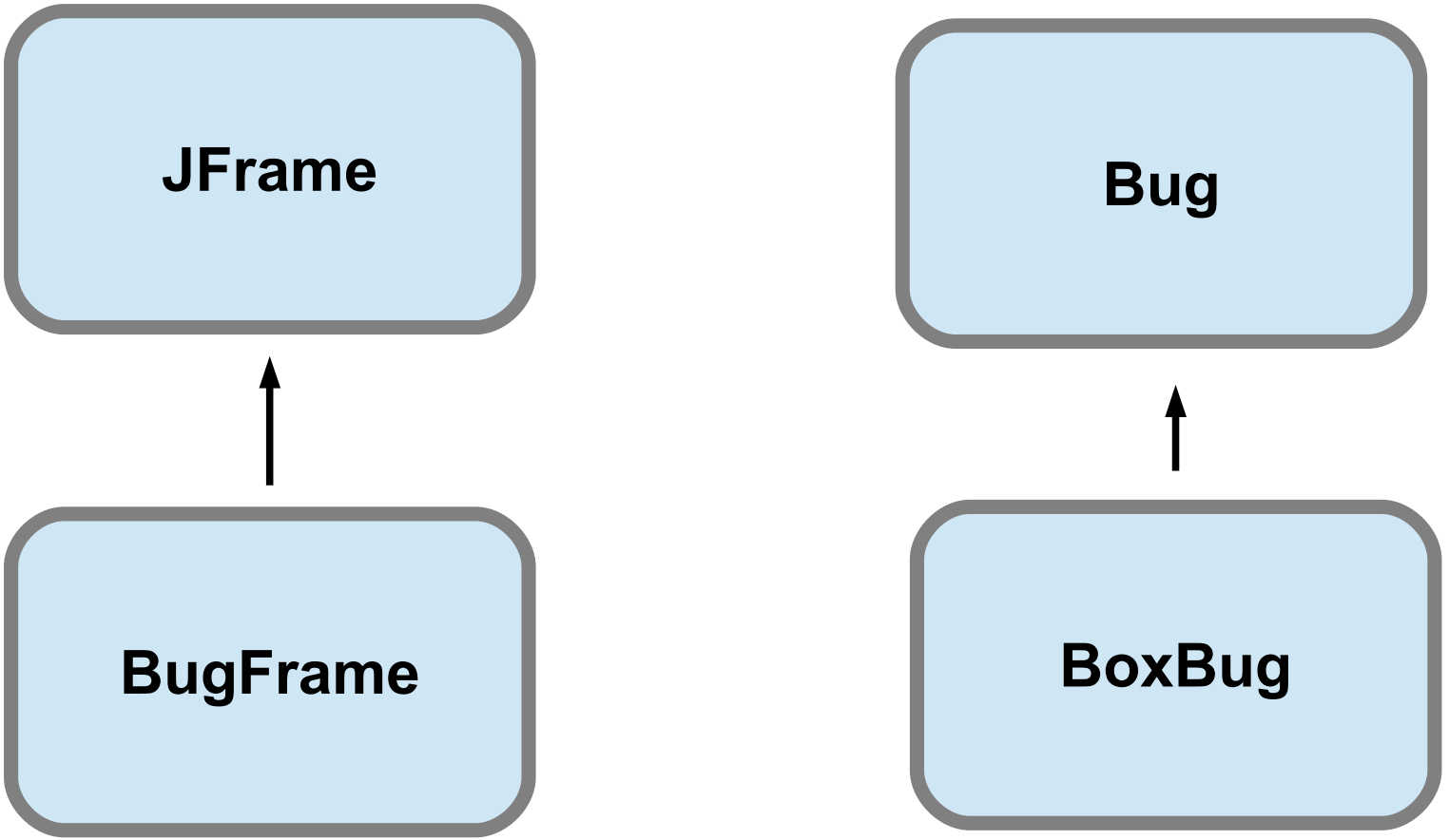
Inheritance and Polymorphism


JFrame extends a class too...
JFrame
GameFrame


Person
Student
Teacher
Exchange Student


Screen
ColorScreen
ColorScreenPlus


JFrame
Bug
BugFrame
BoxBug

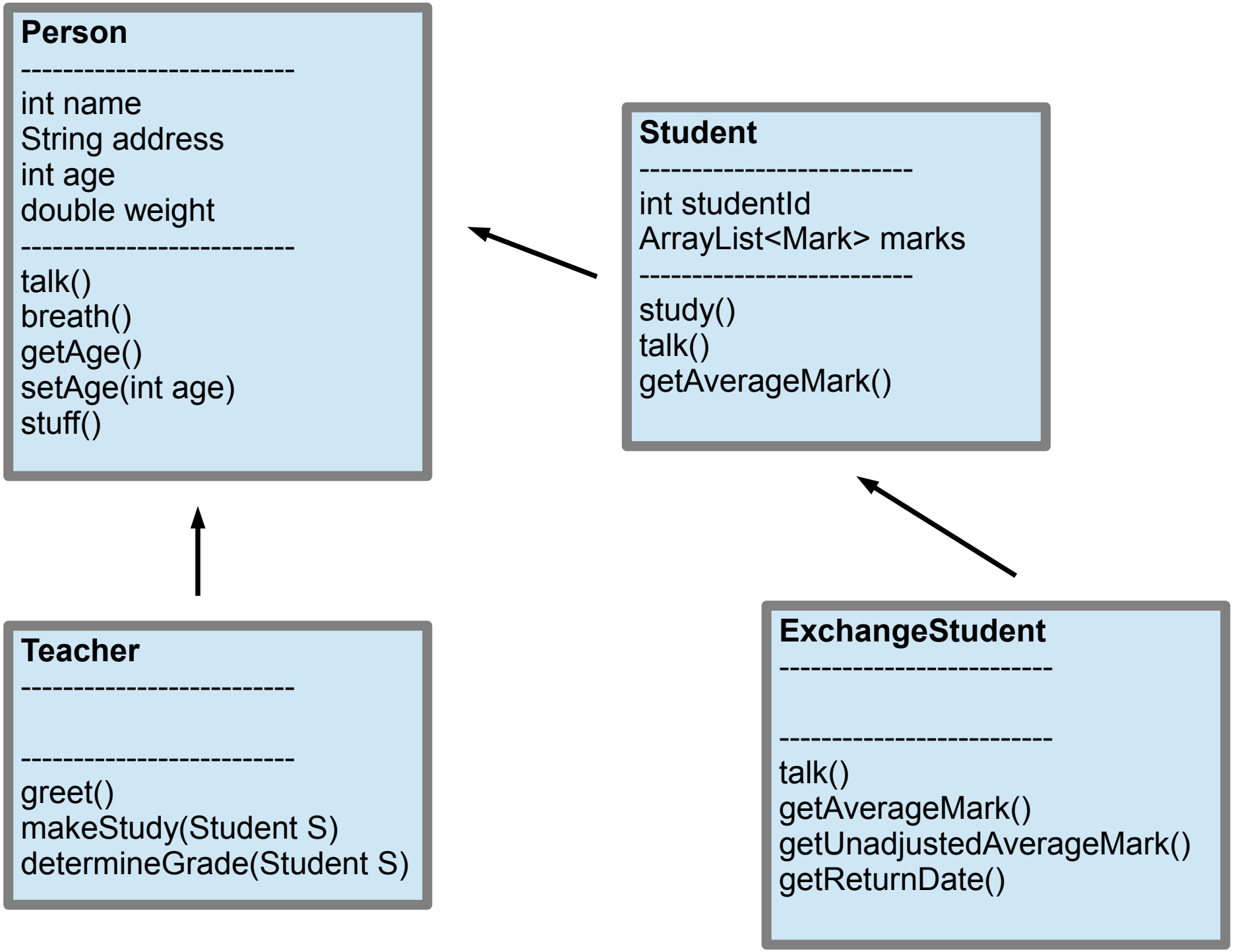
Overrided methods in the constructor of SUPER CLASSES!

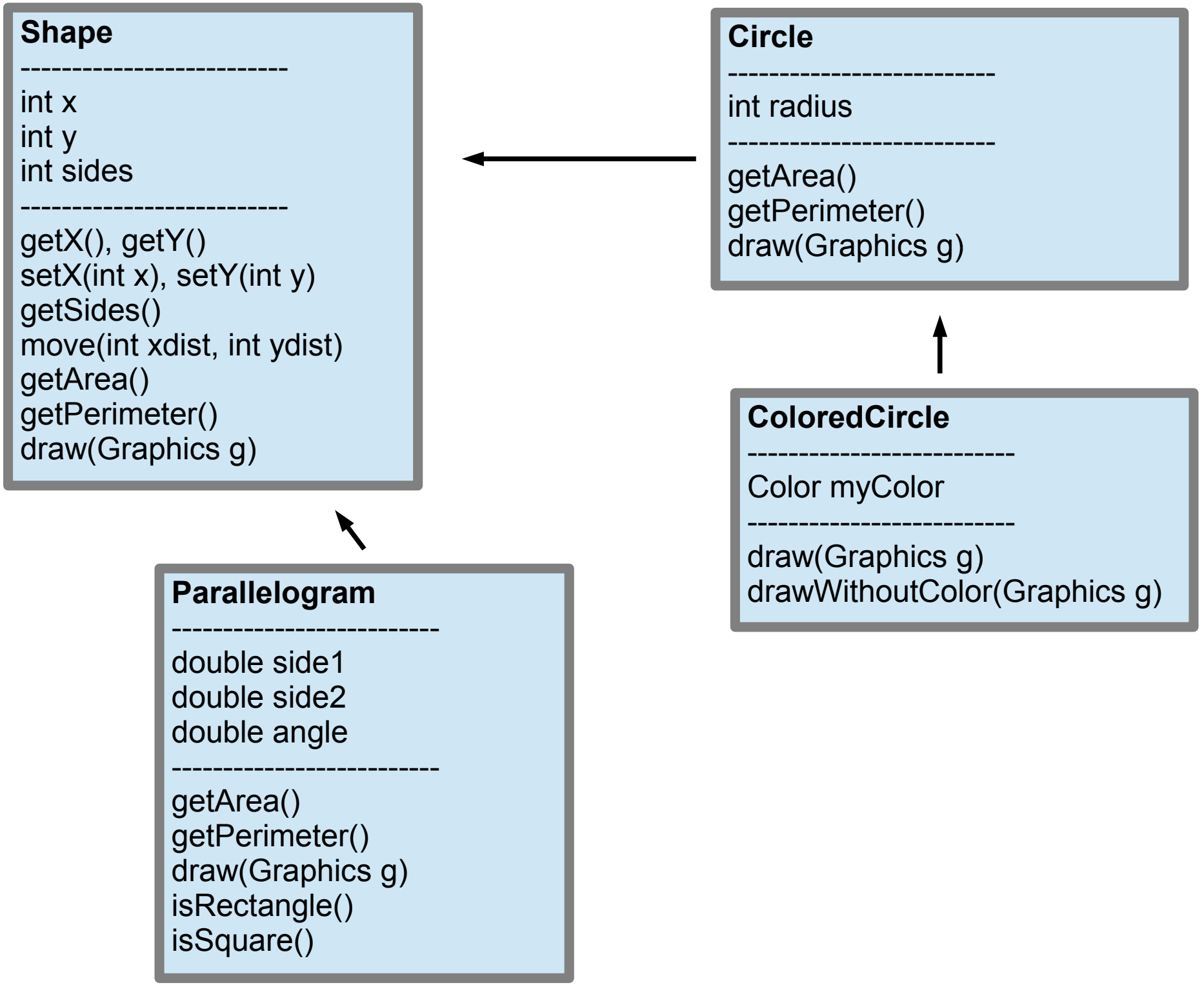
Polymorphism

Polymorphism is the ability of
an object to take on many forms.

The most common use of polymorphism in OOP occurs
when a parent class reference is used to refer
to a child class object.

Any Java object that can pass more than one
IS-A test is considered to be polymorphic.


Person
int name
String address
int age
double weight
talk()
breath()
getAge()
setAge(int age)
stuff()
Student
int studentId
ArrayList<Mark> marks
study()
talk()
getAverageMark()
Teacher
greet()
makeStudy(Student S)
determineGrade(Student S)
ExchangeStudent
talk()
getAverageMark()
getUnadjustedAverageMark()
getReturnDate()

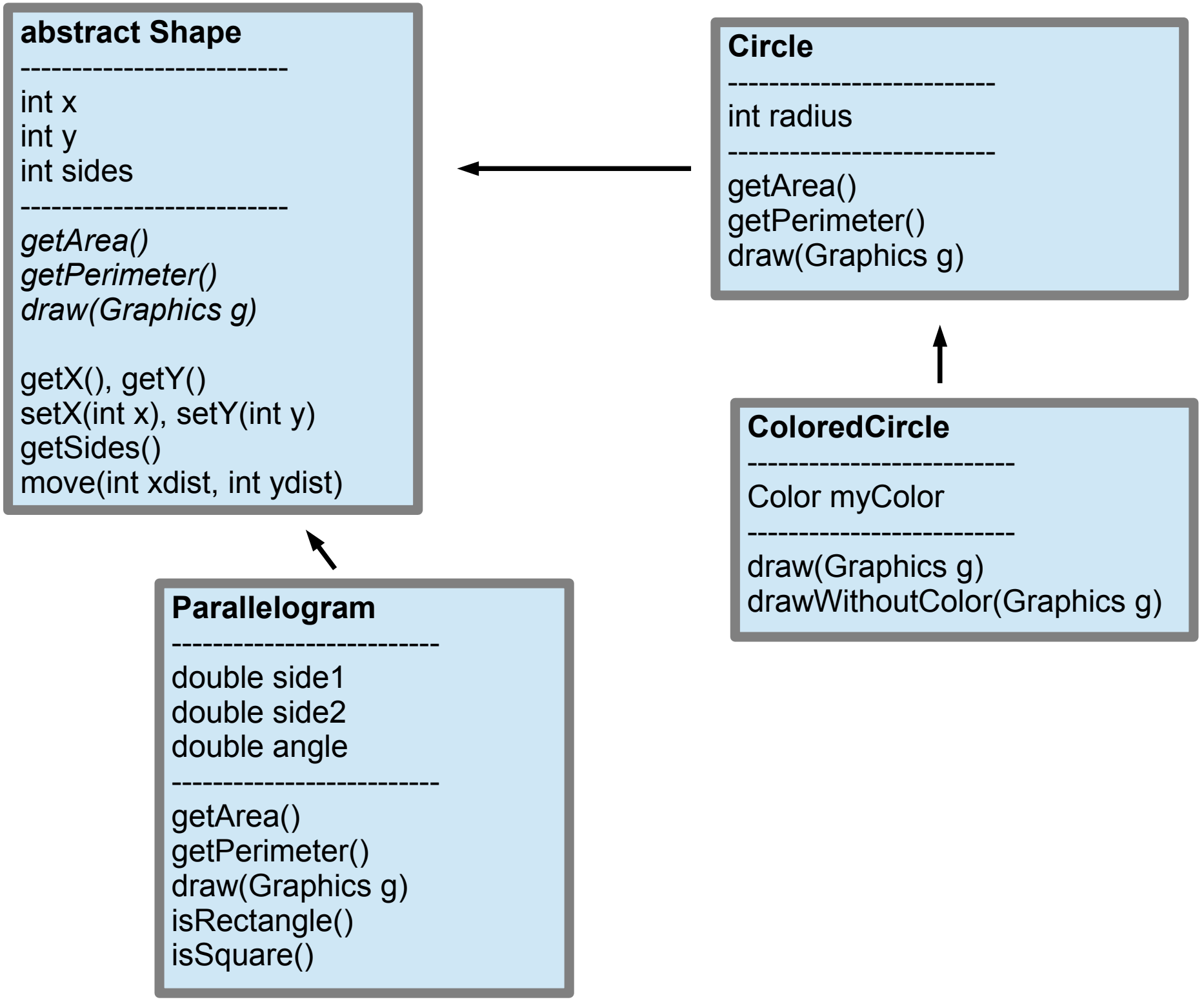

Shape
int x
int y
int sides
getX(), getY()
setX(int x), setY(int y)
getSides()
move(int xdist, int ydist)
getArea()
getPerimeter()
draw(Graphics g)
Circle
int radius
getArea()
getPerimeter()
draw(Graphics g)
ColoredCircle
Color myColor
draw(Graphics g)
drawWithoutColor(Graphics g)
Parallelogram
double side1
double side2
double angle
getArea()
getPerimeter()
draw(Graphics g)
isRectangle()
isSquare()

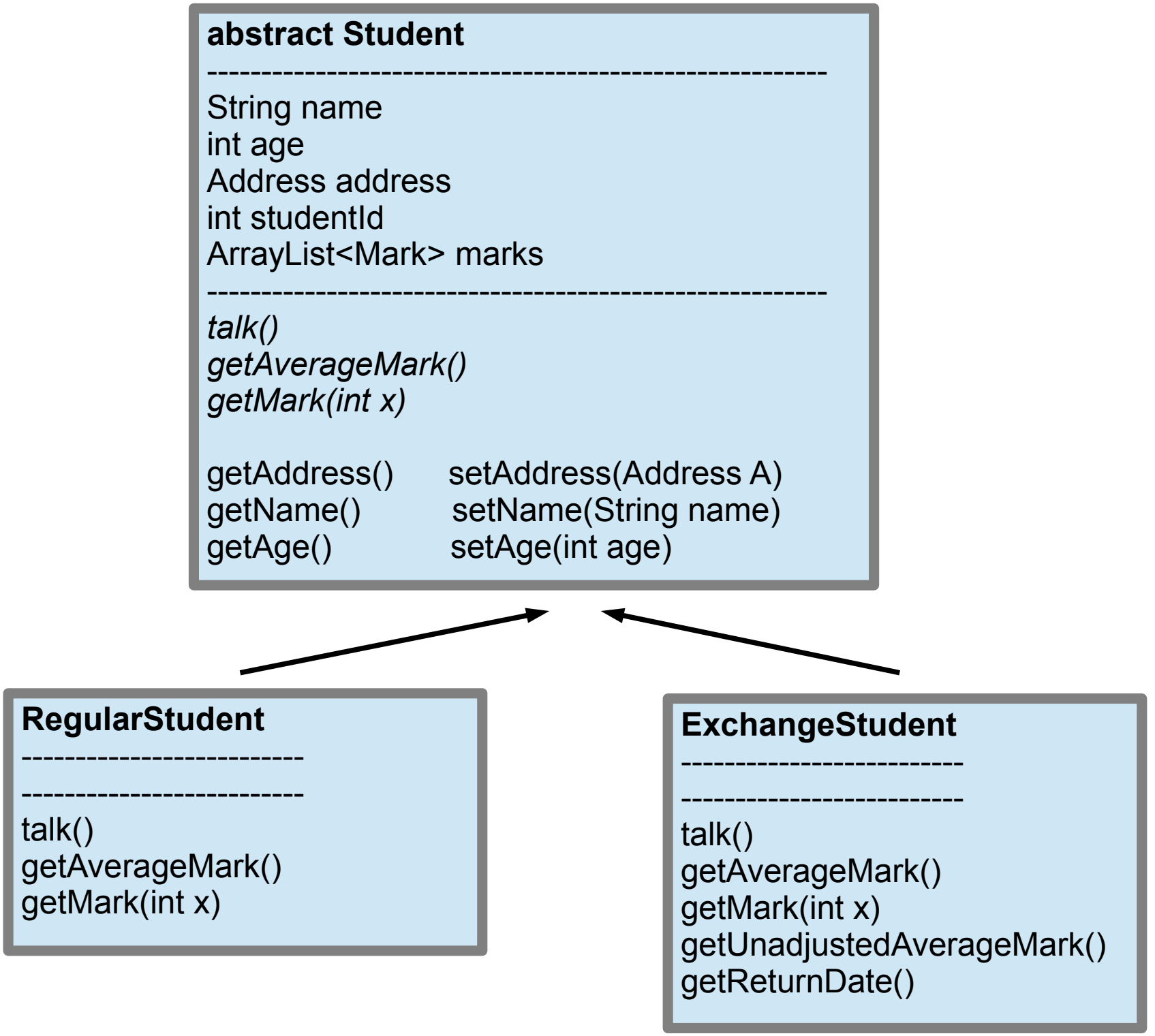
Abstract Classes

can't instantiate an instance of an abstract class
(used as a class to build off of, complete, or add on to)

good way to share fields, methods, and behavior that is common to all subclasses, no copy and pasting code

makes it clear that subclasses will have different implementations of some of the methods (the abstract ones)


abstract Shape
int x
int y
int sides
getArea()
getPerimeter()
draw(Graphics g)
getX(), getY()
setX(int x), setY(int y)
getSides()
move(int xdist, int ydist)
Circle
int radius
getArea()
getPerimeter()
draw(Graphics g)
ColoredCircle
Color myColor
draw(Graphics g)
drawWithoutColor(Graphics g)
Parallelogram
double side1
double side2
double angle
getArea()
getPerimeter()
draw(Graphics g)
isRectangle()
isSquare()


abstract Student
String name
int age
Address address
int studentId
ArrayList<Mark> marks
talk()
getAverageMark()
getMark(int x)
getAddress()
setAddress(Address A)
getName()
setName(String name)
getAge()
setAge(int age)
RegularStudent
talk()
getAverageMark()
getMark(int x)
ExchangeStudent
talk()
getAverageMark()
getMark(int x)
getUnadjustedAverageMark()
getReturnDate()

Interfaces

In the Java programming language, an interface is a reference type, similar to a class, that can contain only constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces.

A class may implement multiple interfaces.

Differences between Interface and Abstract Class

The key technical differences between an abstract class and an interface are:

Abstract classes can have constants, members, method stubs (methods without a body) and defined methods, whereas interfaces can only have constants and methods stubs.

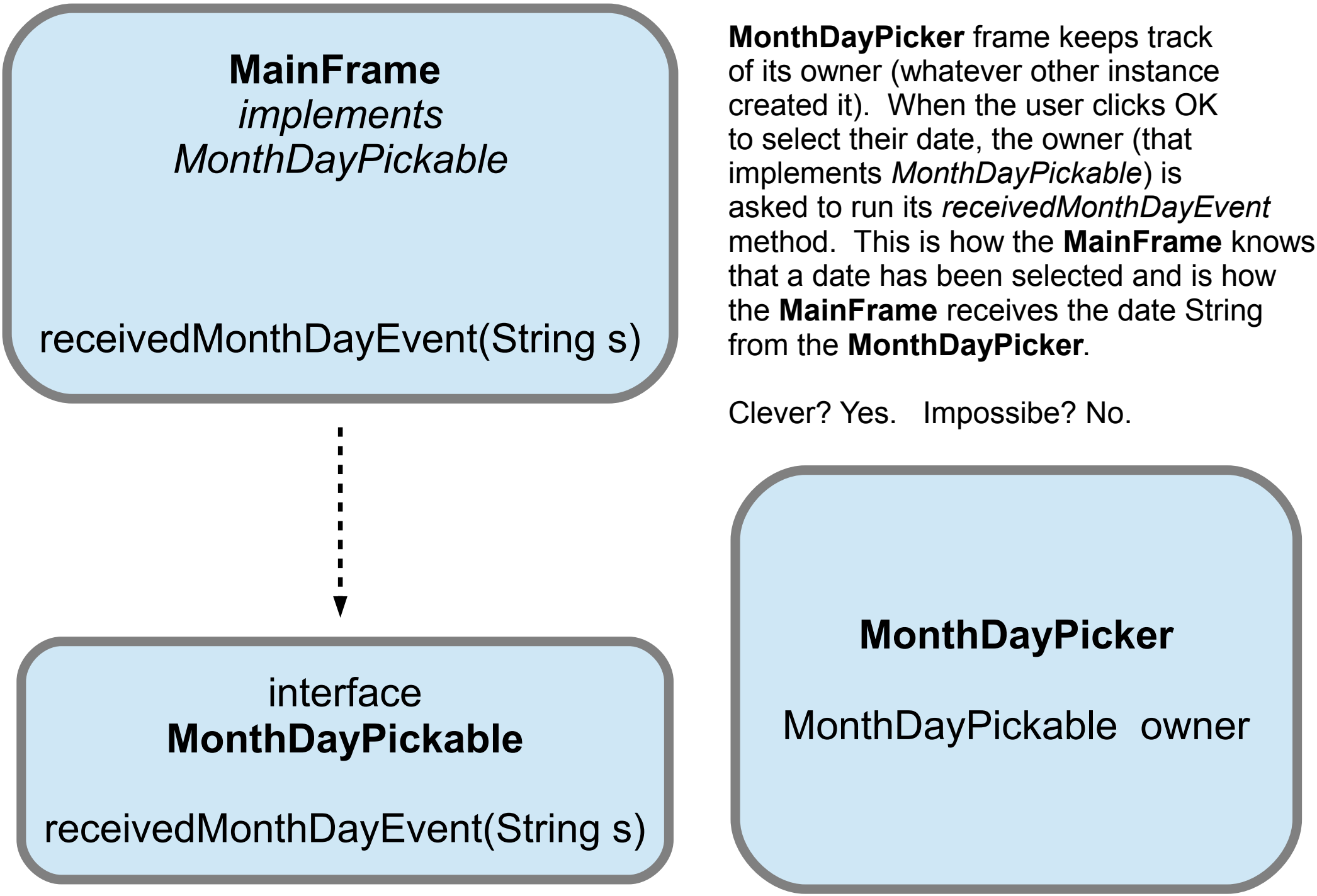
Methods and members of an abstract class can be defined with any visibility, whereas all methods of an interface must be defined as public (they are defined public by default).

When inheriting an abstract class, a concrete child class must define the abstract methods, whereas an an abstract class can extend another abstract class and abstract methods from the parent class don't have to be defined.

Similarly, an interface extending another interface is not responsible for implementing methods from the parent interface. This is because interfaces cannot define any implementation.

A child class can only extend a single class (abstract or concrete), whereas an interface can extend or a class can implement multiple other interfaces.

A child class can define abstract methods with the same or less restrictive visibility, whereas a class implementing an interface must define the methods with the exact same visibility (public).



MonthDayPicker frame keeps track of its owner (whatever other instance created it). When the user clicks OK to select their date, the owner (that implements *MonthDayPickable*) is asked to run its *receivedMonthDayEvent* method. This is how the **MainFrame** knows that a date has been selected and is how the **MainFrame** receives the date String from the **MonthDayPicker**.

Clever? Yes. Impossibe? No.

Interface: **Comparable**

just implement the method

public int compareTo(Object O)

return a number larger than zero if the calling instance is 'larger' than the argument, zero if they are equal, and less than zero if the calling instance is 'smaller'. You decide which propert or properties to use in your comparison!

Benefits to Comparable?

Popular interface that programmers will know how to use

Allows instances or lists of instances of your class to be used in methods that have been designed to work with Comparable types, like the Collections class or any other code that uses Comparable types.