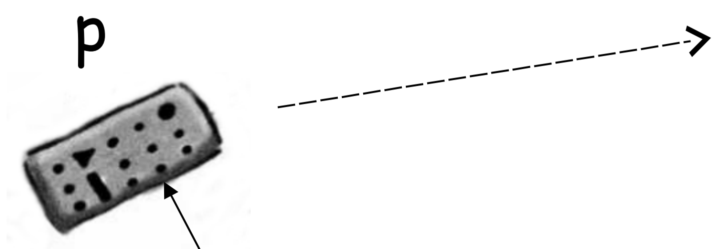
Variables as Remote Control

A useful memory aid

used in *Head First Java*

A Variable is a Reference

$$\underbrace{\text{Person p}}_{\text{a \textit{reference} for sending commands to object}} = \underbrace{\text{new Person( )}}_{\text{object}}$$



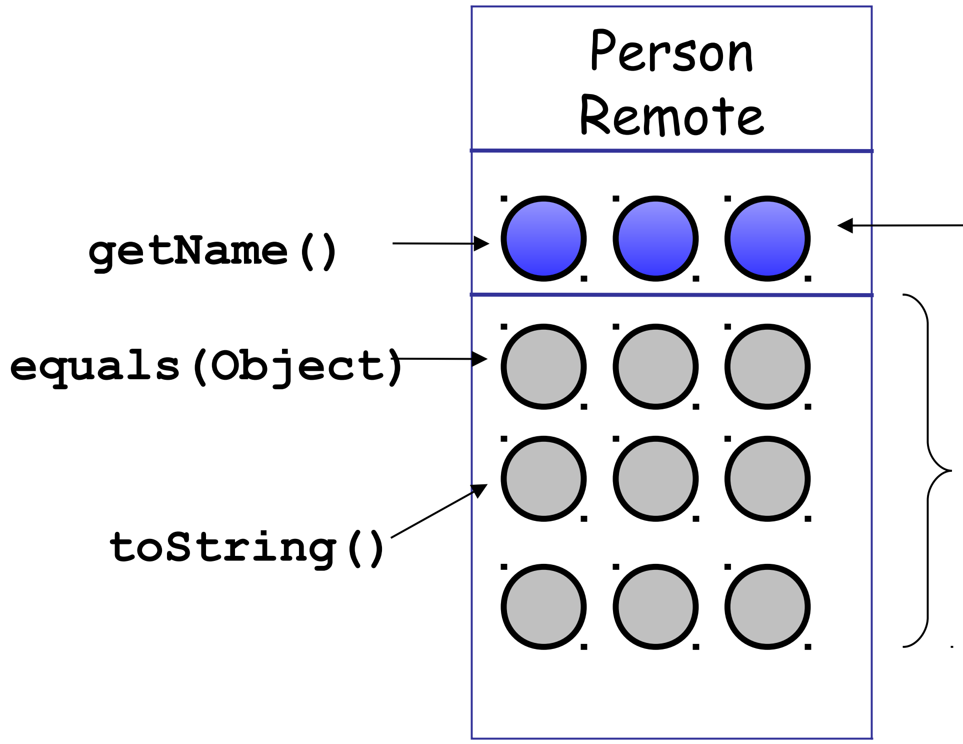
buttons on remote control are methods

Person
#clone()
equals(Object)
finalize()
getClass()
hashCode()
toString()
getName(): Str
getBirthday()

The Compiler decides what Buttons

Person p

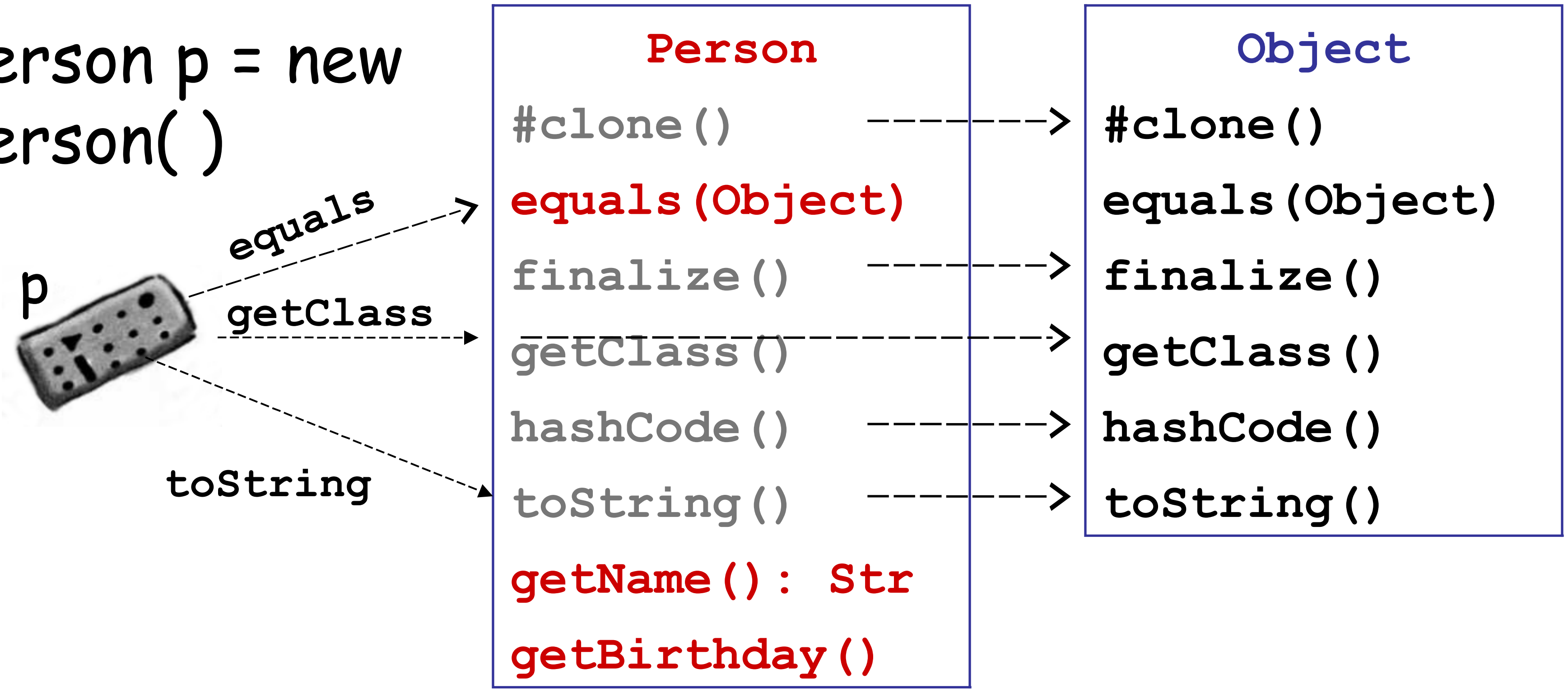
Compiler uses the *declared type* (Person) of a variable to decide what buttons (methods) it has.



Buttons inherited from Object. Every remote control has these.

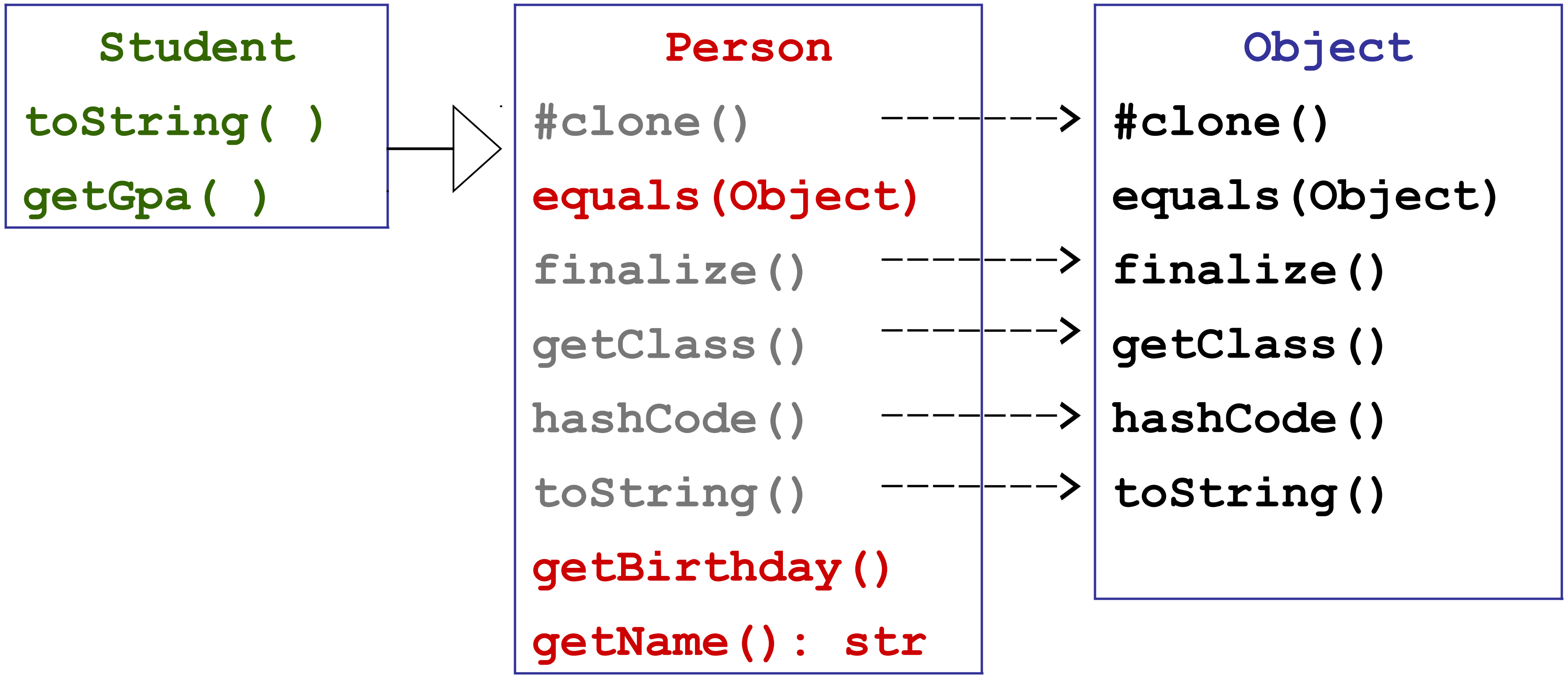
Invoking Methods

Person p = new Person( )



At runtime, JVM invokes method of actual object.
If a class *overrides* a method, the override is used.

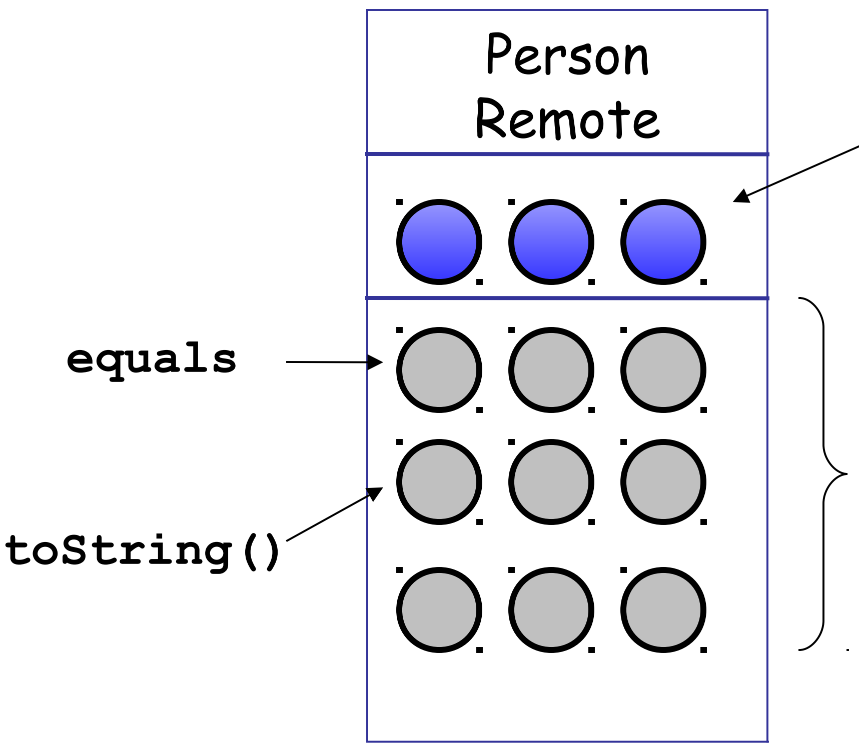
Student extends Person



```
class Student extends Person {
    public double getGpa() { . . . }
    public String toString( ) { . . . }
```

What Buttons Does `p` Have?

Person p = new Student( );



What buttons go here?

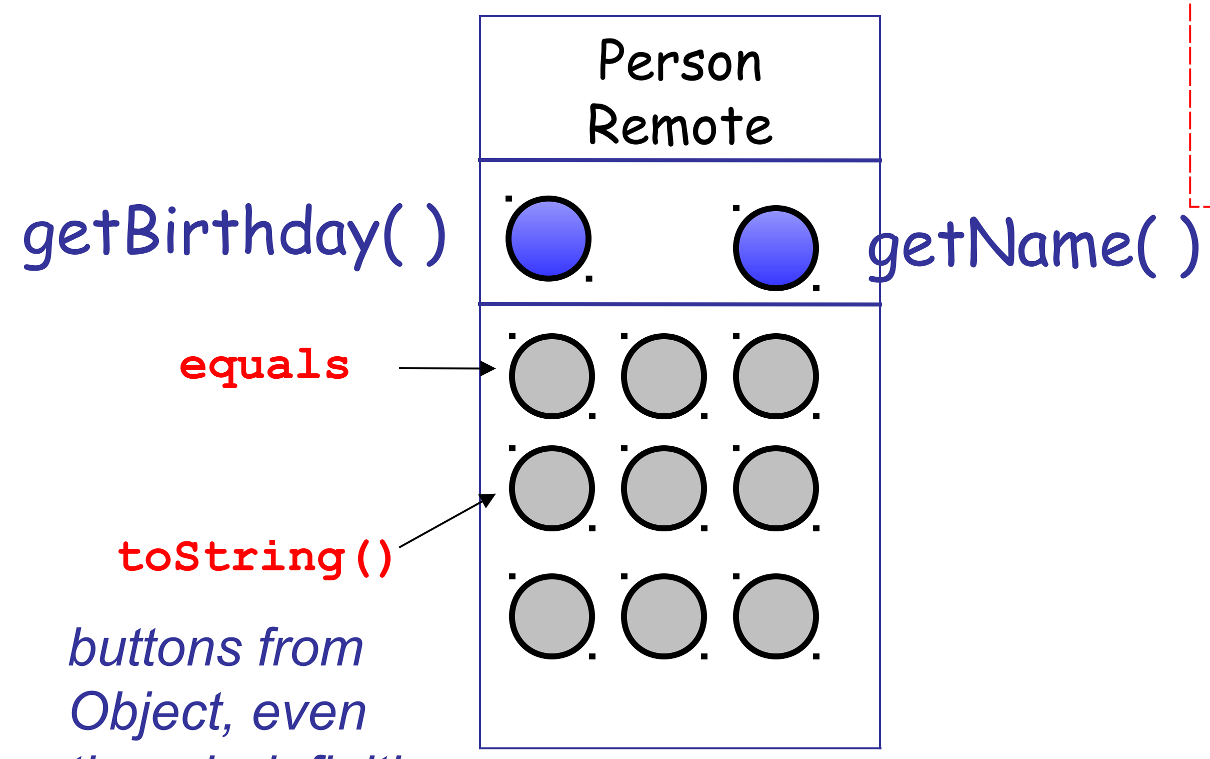
Buttons inherited from Object. Every remote control has these.

What Buttons Does `p` Have?

```
Person p = new Student( );
```

Student has a getGpa method.

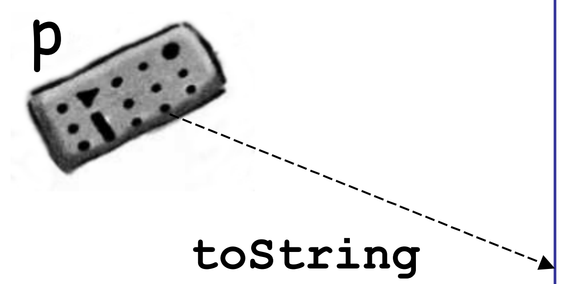
Why is there <u>no</u> getGpa button?



buttons from Object, even though definition is changed.

Invoking toString()

Person p = new Student( )

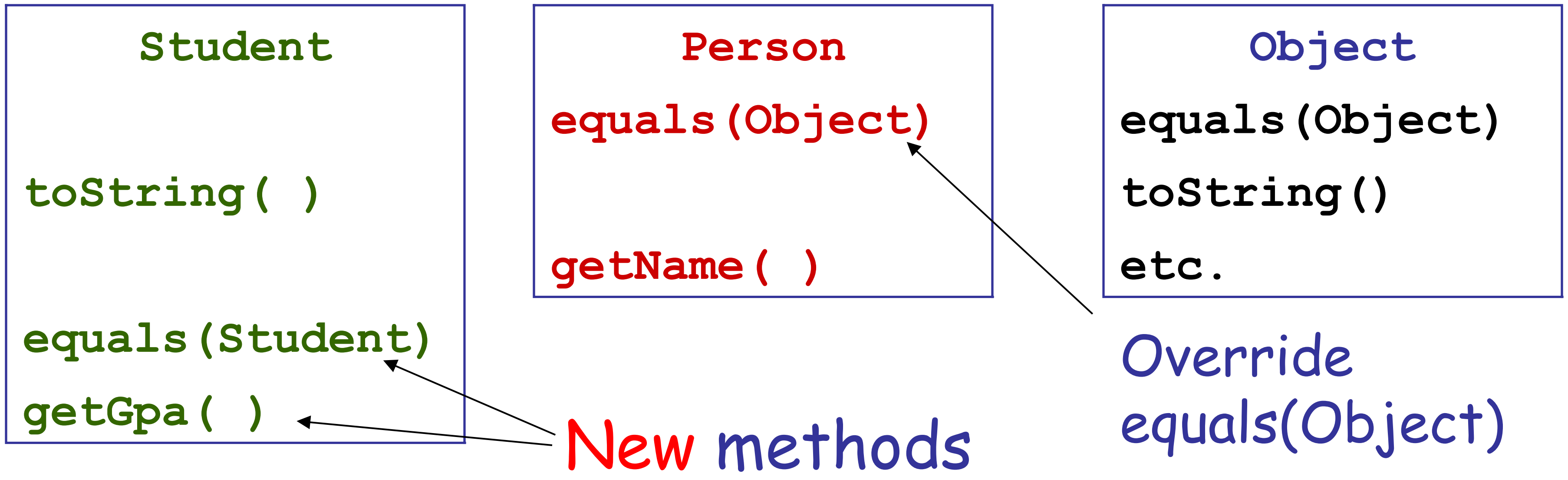


```
Student
#clone()
equals(Object)
finalize()
getClass()
hashCode()
toString()
getName(): Str
getBirthday()
```

```
Object
#clone()
equals(Object)
finalize()
getClass()
hashCode()
toString()
```

Student defines its own **`toString()`**, so the remote calls Student `toString`. It *overrides* Object.toString().

Method Signature includes Parameters



```
class Student extends Person {
    public boolean equals( Student s ) // BAD IDEA
    public String toString( )          // Override OK
```

Which equals( ) is called?

```
Student

toString( )

equals(Student)
```

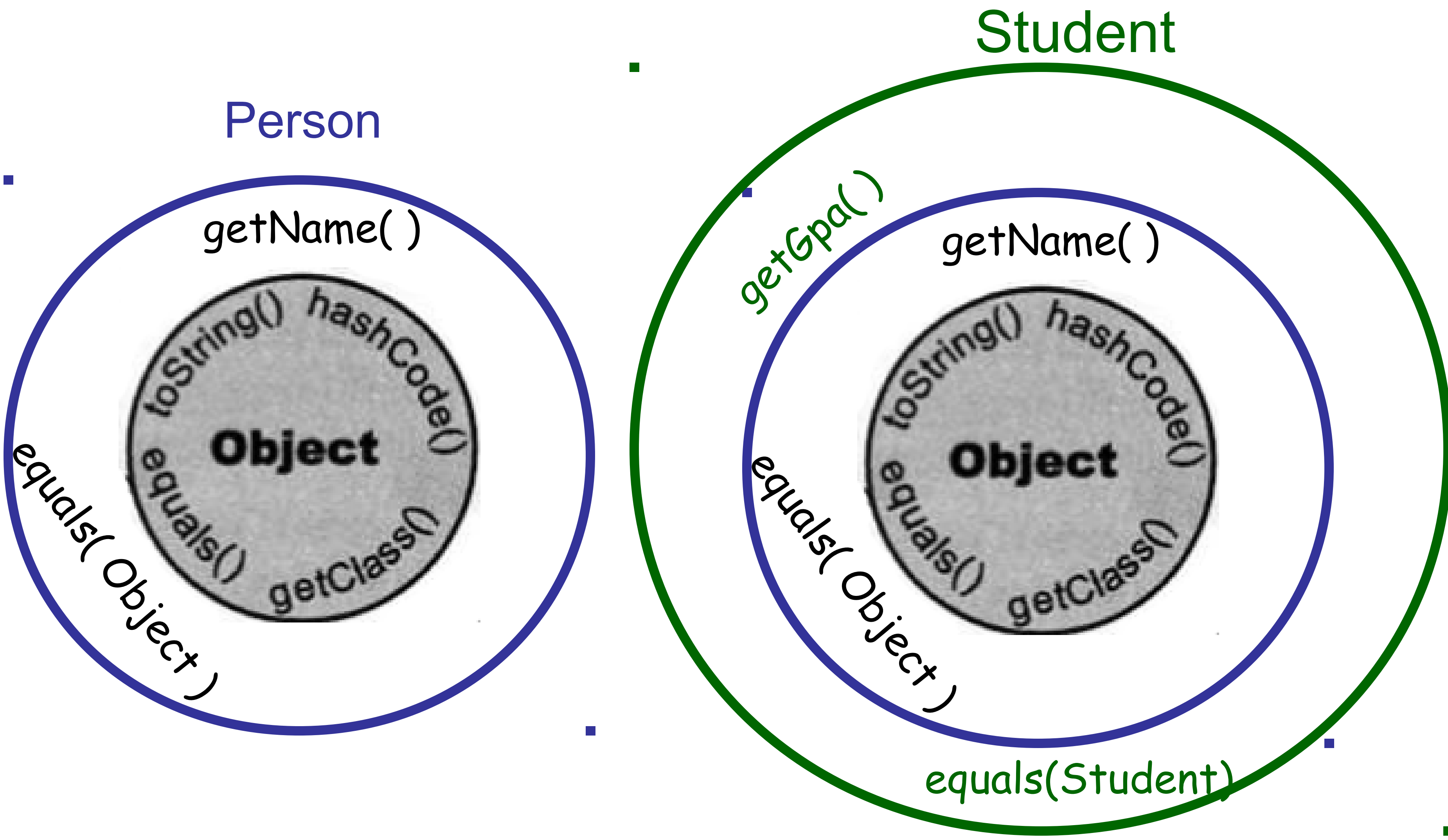
```
Person
equals(Object)

getValue( )
```

```
Object
equals(Object)
toString()
etc.
```

```
Student a = new Student();
Person b = new Student( );
//1.
b.equals( a )
//2.
a.equals( b )
```

Draw the remote control !

Another view of Inheritance

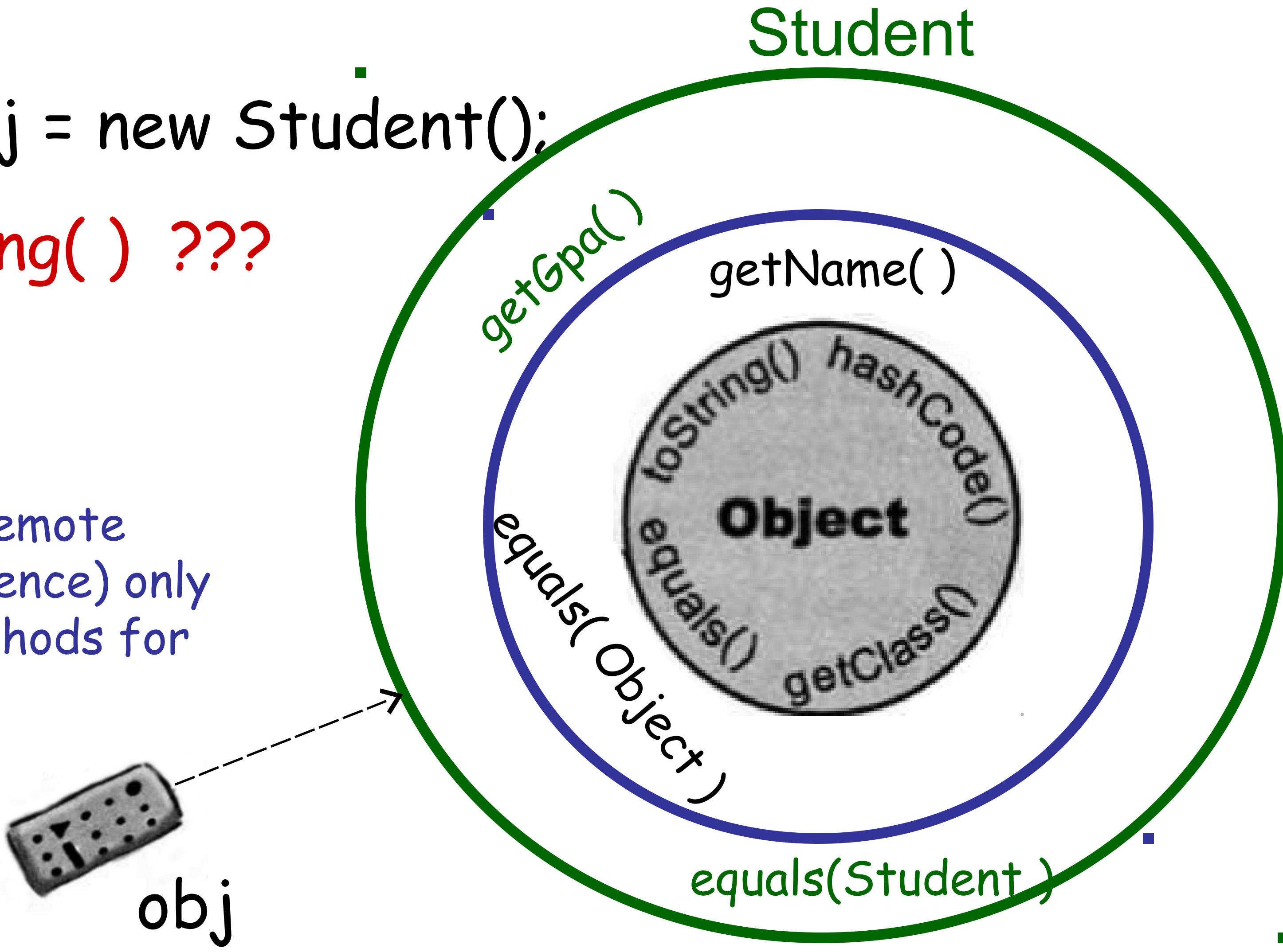


Object References

```
Object obj = new Student();
```

```
obj.toString( ) ???
```

An "Object" remote control (reference) only knows the methods for object.



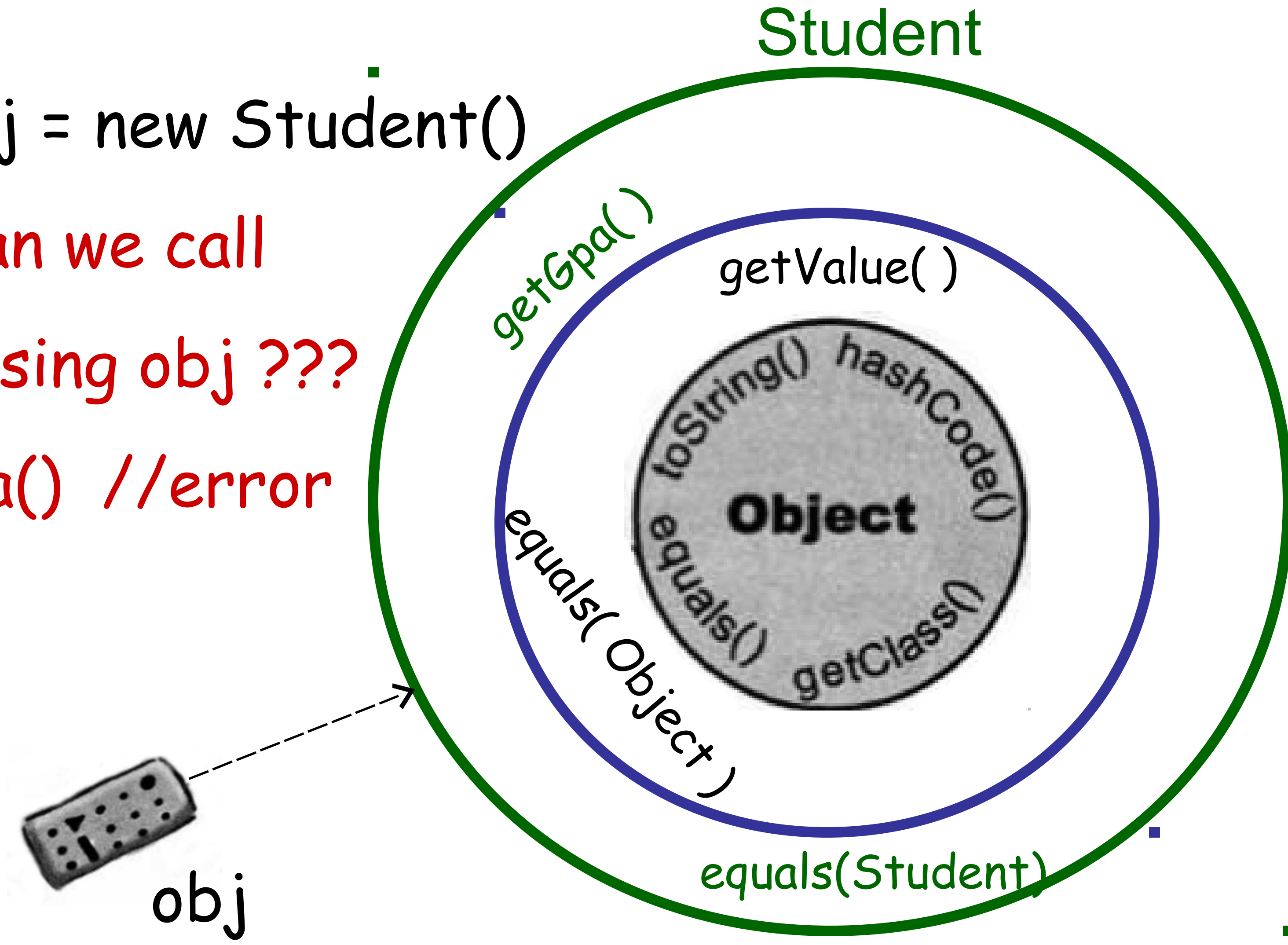
How to Access the *Real* object

Object obj = new Student()

??? how can we call

getGpa() using obj ???

obj.getGpa() //error



Solution: use a *cast*

```
// "Object" remote (reference) only has buttons
// for methods of Object class
Object obj = new Student();
// Cast it to a "Student" reference (remote).
Student s = (Student) obj;
// "Student" remote (reference) has all
// the methods of Student class.
s.getGpa( );    // OK
((Student)obj).getGpa( )
```