Another inheritance example

Let's define a Student as a kind of Person. First, what's a Person?

```
class Person {
  private String name;
  private String birthdate;
  public Person (String name, String bdate) {
    this.name = name;
    this.birthdate = bdate;
  }
  public String getName () { return name; }
  public String toString() {
    String result = "Name: " + name;
    result += "\nBirthdate: " + birthdate;
    return result;
  }
}
```

An array of Persons

using Person in main( ):

```
  int size = 10;
  Person[] list = new Person[size];
  for (int i = 0; i < list.length; i++) {
    String name = in.readLine();
    String bdate = in.readLine();
    list[i] = new Person(name, bdate);
  }

  for (int i = 0; i < list.length; i++)
    System.out.println(list[i]);
```

A Student is a kind of Person

```
class Student extends Person {
  private String stunum;
  public Student (String name, String bdate,
      String stunum) {
    // How do we set name and birthdate?
    this.stunum = stunum;
  }
  public String getStudentNumber () {
    return stunum;
  }
  public String toString () {
    String result = ...; // same as in Person
    result += "\nStudent number: " + stunum;
    return result;
  }   }
```

How is Student related to Person?

This works:

```
  Student stu = new Student("Jim", "1900",
      "0101");
  System.out.println(stu.getName());
```

The getName( ) method is ***inherited*** from Person.

What if we want to add eye colour as an attribute of Persons?
We'll need to change toString( ):

```
  result += "\nEye colour: " + eyeColour;
```

– and the same in Student (and every other child class).
We can use the keyword ***super*** to refer to Person's methods:

```
 String result = super.toString();
```

Two uses of "super"

```
class Student extends Person { ...
  public Student (String name, String bdate,
      String stunum) {
    super(name, bdate); // calls Person's constructor
    this.stunum = stunum;
  } ...

  public String toString () {
    String result = super.toString();
      // calls Person's toString( )
    result += "\nStudent number: " + stunum;
    return result;
  }
}
```

Polymorphism

A Student is also a Person, and has a getName( ) method:

```
Student s = new Student("Jim","1900","0101");
System.out.println(s.getName());
```

A Student has its own toString( ) method:

```
System.out.println(s.toString());
```

– prints name, birthdate, ***and*** student number.

But …

```
Person p = new Student("Jim","1900","0101");
System.out.println(p.toString());
```

This also prints name, birthdate, and student number.

Polymorphism

Java finds the method belonging to the actual object.

Sometimes you need a cast

This doesn't work:

```
Person p = new Student("Jim","1900","0101");
System.out.println(p.getStudentNumber());
```

Why not?

Instead, you need:

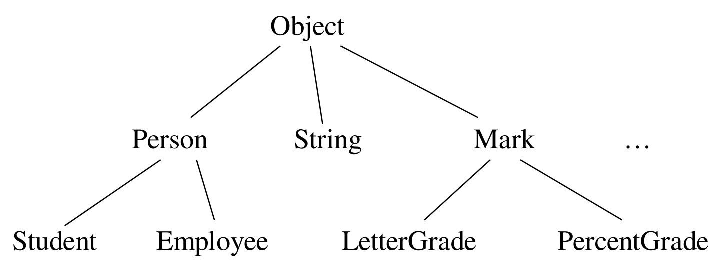
```
System.out.println(  ((Student)p) .
                  getStudentNumber()  );
```

The class Object

All objects are Objects.

- That is, all classes are subclasses of Object.
  - Omitting "extends" means "extends Object".

The is a single "family tree" of classes:



Why Object?

Because the class Object is the ancestor of all other classes:

- We can write methods that accept any object as parameter.
- There are methods that work for any object.
  - toString( ), equals( ).

Not all languages work like this.

- Specifically, C++ (Java's immediate parent) does not have a single class that is the ancestor of all others.
  - C++ allows a class to extend multiple "parent" classes.
  - Java achieves the same purposes differently.

Object-orientedness: a review and clean-up

Our example:

```
class Mark {             // BASE CLASS
  public int intValue() { ... }
}
class PercentGrade extends Mark { // CHILD
  public int intValue () { return mark; }
}
class LetterGrade extendsMark { // another
  public int intValue () {
    if (...) return 95; ...
  }
}
```

Method overriding

The intValue( ) methods in the children ***override*** the one in Mark.

```
  Mark m = new PercentGrade("75");
  System.out.println(m.intValue());
```

– uses PercentGrade's intValue( ).

Polymorphism

The intValue( ) methods in the children ***override*** the one in Mark.

```
Mark m;
if (...)
  m = new PercentGrade(...);
else
  m = new LetterGrade(...);
System.out.println(m.intValue());
```

In the last line, ***we don't know which intValue( ) method is called!*** That's polymorphism.

Inheritance

Sometimes the parent's definition is just fine.

```
class Mark {
  public int intValue () { ... }
  public double gradePoint () [
    int m = this.intValue(); // polymorphism!
    if (m >= 85) return 4.00;
    if (m >= 80) return 3.70;
    ... }  }  // end of Mark class
```

Elsewhere:

```
  LetterGrade l = new LetterGrade("B");
  System.out.println(l.gradePoint());
```

The gradePoint( ) method is ***inherited*** from Mark by LetterGrade.