A problem when you don't know the exact type

Suppose we have an array of Persons:

```
Person[] list = new Person[10];
// ... give values to list's elements
// Some are Persons, and some Students.

// Print student numbers.
for (int i = 0; i < list.length; i++)
  System.out.println(list[i].getStuNum());
```

The last line doesn't work: why not?

The **instanceof** operator

We can only print the student number if it's a Student:

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

It ***still*** doesn't work!

The compiler isn't too bright

We have to reassure the compiler:

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

Now it's finally OK.

Do I have to do this stuff all the time?

No, mostly it's pretty straightforward.
Casts are uncommon, and instanceof is pretty unusual.
Here's a more typical example:

```
// Working with a drawing
Shape[] drawing = new Shape[/* ?? */];

class Shape {
  public void draw (Window w) { ... }
  ...
}
```

Shape is general. Presumably there are specific subclasses.

Kinds of Shapes

```
class Rectangle extends Shape {
  private double top, bottom, left, right;
  public void draw (Window w) {
    w.drawRect(top,bottom,left,right);
  }
}
class Circle extends Shape {
  private double xCentre, yCentre, radius;
  public void draw (Window w) {
    w.drawOval(xCentre,yCentre,radius,radius);
  }
}
```

And so on, for whatever other shapes we need.

Displaying the entire drawing

```
  for (int i = 0; i < drawing.length; i++)
    drawing[i].draw(currentWindow);
```

Each element of the array "drawing" knows how to draw itself, and we're guaranteed that each of them has a "draw" method, because Shape has such a method, and they're all Shapes.

But we can't write the draw method for the Shape class.

- It's too general.

We need to ***have*** a draw method in Shape, but we must never call it.

(Remember "intValue" for Marks: return -9999!)

Abstract methods and abstract classes

An ***abstract method*** is a method that has no implementation:

```
  abstract public void draw (Window w);
```

Now we have to announce that the class Shape is also abstract:

```
abstract class Shape {
  abstract public void draw (Window w);
}
```

Any child class of Shape has a draw method, because Shape does.
All the child classes of Shape must override draw( ).

Abstract method & class motivations and rules

Make a method abstract when:

- The method is required in any actual object.
- You can't say how to implement it in general.

Rules about abstract methods and classes:

- If any method in a class is abstract, the class must be abstract.
- An abstract class may have non-abstract methods.
- An abstract class cannot be instantiated.
  - but it may have constructors.
- Subclasses of an abstract class that want to be non-abstract must override ***all*** inherited abstract methods.

The Mark class again

```
abstract class Mark {
  abstract public int intValue ();
  public double gradePoint () [
    int m = intValue();
      // We can call intValue, because any actual object
      // belongs to a class that has implemented the method.
    if (m >= 85) return 4.00;
    if (m >= 80) return 3.70;
    …
  }
}
```

Classes for a quiz

```
class Csuper {
  public String name;
  public Csuper (String name) {
    this.name = name; }
  public String toString () {
    return name + " -- a super";}
}
class Csub extends Csuper {
  public Csub (String name) { super(name); }
  public String toString () {
    return name + " -- a sub";}
  public String supersToString () {
    return super.toString();}
}
```

The main( ) method for the quiz

```
Csuper xsuper = new Csuper("xsuper");
Csub xsub = new Csub("xsub");
Csuper xhuh = new Csub("xhuh");

System.out.println (xsuper.toString());
System.out.println
  (((Csub)xsuper).supersToString());
System.out.println (xsub.toString());
System.out.println (xsub.supersToString());
System.out.println (xhuh.toString());
System.out.println
  (((Csub)xhuh).supersToString());
```