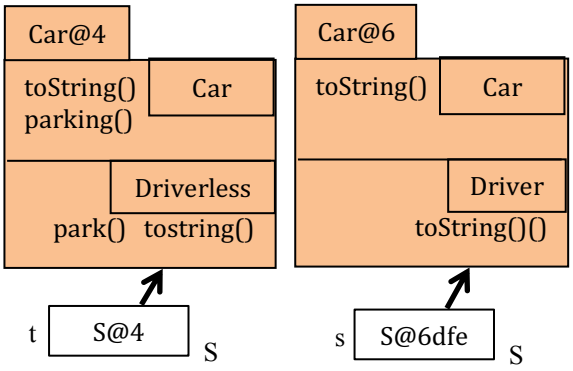
An example in which instanceof should be used

In discussing function equals, we showed a case in which function getClass should be use instead of operator instanceof. Here is an example that requires instanceof and not getClass.

Car manufacturer Ferd is preparing for the day when some cars will be driverless. Ferd is building code that will be embedded in each car. Ferd has built a class Car that expects exactly two subclasses: Driver and Driverless, objects of which are shown to the right. The two subclasses are needed because, while both kinds of cars have a lot of similarities, which will be housed in class Car, they obviously have big differences —e.g. a driverless car may not even have a steering wheel. Ferd hired two other firms to write the subclasses.

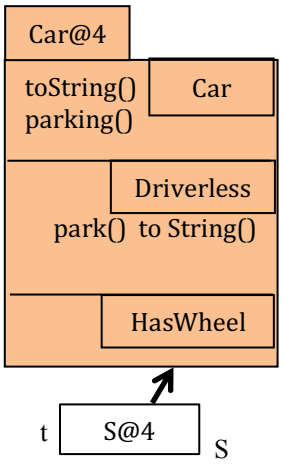


A driverless car has software that will actually park the car.

To park a car, procedure parking in class Car has to call method park in class Driverless. But procedure parking has to determine whether the car is driverless and only then call park. This method appears to the right.

```
/** If this is a driverless car,
  * call park(). */
private void parking() {
   if (this.getClass() ==
                Driverless.class) {
      (Driverless) park();
}
```

But this doesn't work!



Manufacturer Ferd did not know that the firm that built class Driverless felt it useful to have subclasses of Driverless, and one of them is shown to the left. It's for a car that has a steering wheel.

In this object, this.getClass() is an object that describes class HasWheel, not class Driverless. The correct way to write procedure park is shown to the right. It uses operator instanceof.

THIS METHOD IS CORRECT

```
/** If this is a driverless car,
  * call park(). */
private void parking() {
   if (this instanceof Driverless) {
      (Driverless) park();
}
```

Moral of the story: To determine whether an object ob has a partition named C, use

ob instanceof C

To determine the class of object ob, use

ob.getClass()