Polymorphism

- Objectives - when we have completed this set of notes, you should be familiar with:
  - Defining polymorphism and its benefits
  - Using inheritance to create polymorphic references
  - Using interfaces to create polymorphic references

Polymorphism

- The term *polymorphism* literally means "having many forms"
- A *polymorphic reference* can refer to objects of various types
- The method invoked through a polymorphic reference can change from one invocation to the next

Polymorphism

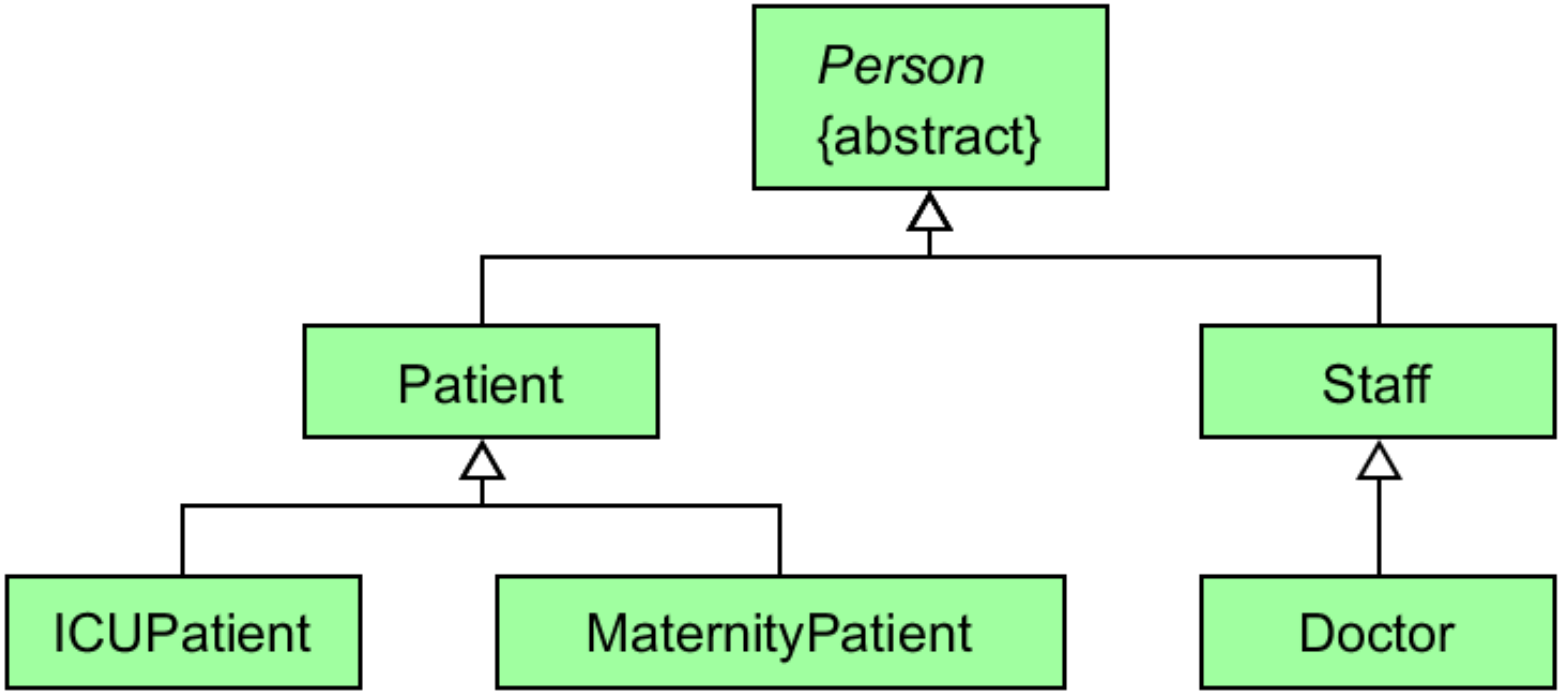
- Consider the following reference variable:

```
Occupation job;
```

- Java allows the variable `job` to reference an `Occupation` object or any other object of a compatible type
- Compatibility can be established by **inheritance** (subclasses) or by using **interfaces**
- Now let's take another look at the Person class hierarchy to see how compatibility is established by inheritance

References and Inheritance

- Consider the following inheritance hierarchy:



(Open the Hospital.gpj project file to generate UML Class Diagram)

References and Inheritance

- An object reference can refer to an object of its class, **or to an object of any subclass**
- For example, the following code is valid:

```
Person p = new Patient("Jane", "Lane", "US", 1970);
p = new ICUPatient("Jake", "Lane", 19, 1980);
Staff s = new Doctor("Joan", "Lane", "Lab", "");
p = s;  // can we do this?
```

- Remember, we have the *is-a* relationship
- Assigning a child object to a parent reference is considered a *widening conversion*

References and Inheritance

- Assigning an parent reference to a child reference can be done also, but it is considered a narrowing assignment and must be done with a cast. In addition, the object referenced by the parent type must be an instance of the child class or one of its subclasses. Example:

```
Staff s = new Doctor("Joan", "Lane", "Lab", "");
Doctor d = (Doctor) s;
```

- The widening conversion is the most useful

Polymorphism via Inheritance

- When an object reference is declared as a parent type, you only have access to the methods in the parent class.
  - The methods specific to the child class can only be accessed using casting.
- For example, you cannot invoke s.setSpecialty() on the s object below even though setSpecialty is defined in Doctor.java. A cast is required since it is unknown to Staff.

```
Staff s = new Doctor("Joan", "Lane", "Lab", "");
((Doctor)s).setSpecialty("Surgeon");
```

Polymorphism via Inheritance

- The *instanceof* operator can be used to determine if an object is an instance of class.
- For example:

```
Doctor d = new Doctor("Joan", "Lane", "Lab", "");
if (d instanceof Doctor) {
   System.out.println("d is a Doctor");
}
if (d instanceof Person) {
   System.out.println("d is a Person");
}
```

Would print the following:

```
d is a Doctor
d is a Person
```

InstanceofExample.java

Polymorphism via Inheritance

- When a method is invoked on object referenced by a superclass variable, although the method must be present in the superclass, the version of the method associated with the subclass is the one that is actually invoked.
- Consider HospitalExample.java
- Note that different versions of getId() are invoked even though all objects in personArray are of type Person.

```
for (Person p : personArray) {
     System.out.println(p.getId());
}
```

Binding

- Consider the following method invocation:

```
obj.toString();
```

- At some point, this invocation is *bound* to the definition of the method that it invokes
- If this binding occurred at compile time, then that line of code would call the same method every time
- However, Java defers method binding until run time -- this is called *dynamic binding* or *late binding*
- Thus, in our Hospital example, the appropriate getID() method was bound while the program was running

Polymorphism via Interfaces

- An interface name can be used as the type of an object reference variable

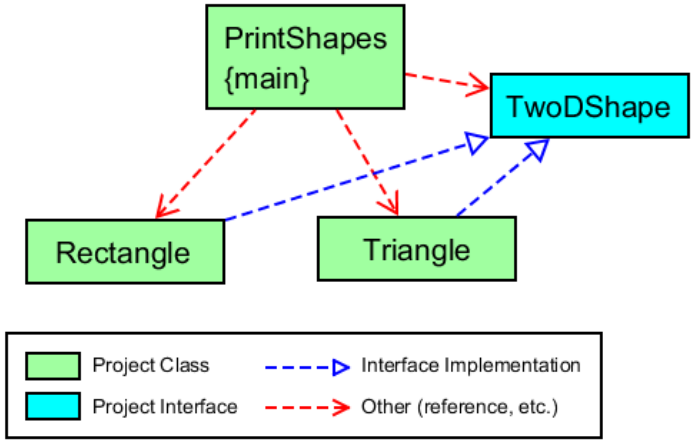
```
Comparable current;
```

- The variable `current` can be used to reference an object of any class that implements the `Comparable` interface
- The version of `compareTo` invoked below depends on the type of object that `current` references

```
int result = current.compareTo(cObj);
```

Polymorphism via Interfaces

- Recall our Rectangle and Triangle classes that implement the TwoDShape interface
- PrintShapes.java provides an example of polymorphism via inheritance



- The variable TwoDShape shape1 can access getNumberSides(), getPerimeter(), and toString(), but what about getClassification()?

```
((Triangle) shape1).getClassification()
```

Polymorphism

- All object references in Java are potentially polymorphic, because all classes inherit from the Object class and all interfaces have implicit abstract methods that match the public methods in the Object class unless these methods are explicitly declared [JLS 9.2]).
- The object class has the following methods (as well as others)
  - clone(), equals(), toString()
- Thus any object reference, regardless of its declaration type, can access the above methods

Polymorphism

- The instanceof operator, in addition to being used to determine if an object is an instance of a class, can be used with a variable of an interface type that references the object

  Consider the following in interactions:

```
Comparable c = "I'm a String";
c instanceof String
true
c instanceof Comparable
true
```