

Polymorphism

Understanding Polymorphism

- property of the object to take many different forms (poly = many, morph = form)
- to access an Java object, we have to have *a reference* pointing to that object
- there are three ways to access the Java object:
 1. using reference with the same type as the object
 2. using reference that is superclass of the object
 3. using interface reference

```
public class Dog {  
    public boolean canRun() { return true; }  
}
```

superclass

```
public interface CanRun { boolean canRanFast(); }
```

interface with one abstract method

```
public class ShihTzu extends Dog implements CanRun {
```

```
    public boolean canRanFast() { return false; }
```

implementation of interface method

```
    public int weight = 6;
```

```
    public static void main(String[] args) {
```

```
        ShihTzu shihTzu = new ShihTzu();  
        System.out.println(shihTzu.weight);
```

creating a reference using
the same type as object

```
        Dog dog = shihTzu;  
        System.out.println(dog.canRun());
```

creating a reference using
superclass type

```
        CanRun canRun = shihTzu;  
        System.out.println(canRun.canRanFast());
```

creating a reference using
interface type

```
    }  
}
```

6
true
false

NOTE: Only one object is created here (ShihTzu) !!

// once you create a new reference, only the members of that reference
// type are accessible via that reference (!!)

```
CanRun canRun = new ShihTzu();
```

=> reference canRun points to ShihTzu object -> OK

```
System.out.println(canRan.weight);
```

=> DOES NOT COMPILE

=> weight is not a member of CanRun interface

```
Dog dog = new ShihTzu();
```

=> reference dog points to ShihTzu object -> OK

```
System.out.println(dog.canRunFast());
```

=> DOES NOT COMPILE

=> method canRunFast() is not a member of Dog class

Object vs. Reference

1. The **type of the object** determines which properties exist within the object in memory.
2. The **type of the reference** to the object determines which methods and variables are accessible to the Java program.

Reference

shihTzu

dog

canRun

Object in memory (ShihTzu)

weight = 6

canRun()

canRunFast()

```
// casting objects
```

```
ShihTzu shihTzu = new ShihTzu();
```

```
Dog dog = shihTzu;
```

=> implicit casting to a supertype, OK

```
ShihTzu secondShihTzu = (ShihTzu) dog
```

=> explicit cast to subtype, OK

```
ShihTzu thirdShihTzu = dog;
```

=> ClassCastException

=> you cannot put larger in smaller without the explicit cast

```
// the compiler doesn't allow casts to unrelated types
```

```
public class Cat {}
```

```
public class Dog {  
    public static void main(String[] args) {  
        Dog dog = new Dog();  
        Cat cat = (Cat) dog; DOES NOT COMPILE  
    }  
}
```

```
// this can be avoided using instanceof operator
```

```
// overriding methods
```

```
class Dog {  
    public int getSpeed() { return 20; }  
    public void printSpeed() { System.out.println(this.getSpeed()); }  
}
```

```
public class GreatDane extends Dog {  
    @Override  
    public int getSpeed() { return 35; }
```

```
    public static void main(String[] args) {  
        new GreatDane().printSpeed();  
    }  
}
```

Which getSpeed() will be used?

Since getSpeed() is overridden in a subclass, **all calls** to this method will be replaced at the runtime!

35

you can always limit polymorphism by making methods final, in which case they can't be overridden in a subclass


```
// hiding methods
class Dog {
    public static int getSpeed() { return 20; }
    public void printSpeed() {
        System.out.println(this.getSpeed());
    }
}

public class GreatDane extends Dog {
    public int getSpeed() {
        return 35;
    }

    public static void main(String[] args) {
        new GreatDane().printSpeed();
    }
}
```

Which `getSpeed()` will be used?

Since `getSpeed()` is static,
it cannot be overridden (only hidden),
therefore the `this.getSpeed()` will
always call `getSpeed()` method
as it is defined in the `Dog` class



20

```
// final example
class Cat {
    protected int age = 5;
    public static boolean isWild() { return false; }
}

public class Tiger extends Cat {
    protected int age = 7;
    public static boolean isWild() { return true; }
    public static void main(String[] args) {
        Tiger dave = new Tiger();
        Cat rave = dave;
        System.out.println(dave.isWild());
        System.out.println(rave.isWild());
        System.out.println(dave.age);
        System.out.println(rave.age);
    }
}
```

```
true
false
7
5
```