Polymorphism

```
::::::::::::::
Speaker.java
::::::::::::::
public interface Speaker
{
    public void speak();
}
::::::::::::::
Dog.java
::::::::::::::
public class Dog implements Speaker
{
  public void speak()
  {
    System.out.println("bow wow wow wow!");
  }
}
::::::::::::::
Poet.java
::::::::::::::
public class Poet implements Speaker
{
  public void speak()
  {
    System.out.println("I think; therefore I am.");
  }

  public void write()
  {
    System.out.println("Once upon a time, ..., the end.");
  }
}
::::::::::::::
Driver.java
::::::::::::::
public class Driver
{
  public static void main(String [] args)
  {
    Speaker guest;

    guest = new Dog();
    guest.speak();

    guest = new Poet();
    guest.speak();

    // guest.write();   will not compile because it does not belong to
    // the interface, so let's cast:
    ((Poet)guest).write();
  }
}
```

```
::::::::::::::
Pet.java
::::::::::::::
/** Pet.java - example of polymorphism
 *  We're going to declare a variable pet of the Animal class.
 *  But 'pet' will assume different dynamic types depending on which
 *  constructor we use.
 */
public class Pet
{
  public static void main(String [] args)
  {
      Animal pet;

      System.out.printf("animal #1...\n");
      pet = new Bird();
      pet.feed();

      System.out.printf("\nanimal #2...\n");
      pet = new Fish();
      pet.feed();

      // Let's make a Penguin.  Even casting it as a Bird doesn't fool the
      // compiler.  It's still a Penguin!
      System.out.printf("\nanimal #3...\n");
      pet = new Penguin();
      pet = (Bird) pet;
      pet.feed();
  }
}


::::::::::::::
Pet.out
::::::::::::::
animal #1...
Just created an animal.
 just created a bird
animal is grateful for the grub!
animal eats like a bird!

animal #2...
Just created an animal.
Frank is grateful for the grub!
Fish need to eat too!

animal #3...
Just created an animal.
 just created a bird
  just created a penguin
Tux is grateful for the grub!
Tux eats like a bird!
Penguins love ice cream.
```