Class 7 Notes
Inheritance

Again:

Principles of Object Oriented Programming

- Encapsulation
- **Inheritance**
- Polymorphism

Inheritance : Builds new classes from existing classes

Inheritance

```
public  class Cat
{
        protected  int weight;              // notice the keyword "protected"

        public Cat()
        {
                weight = 10;
        }
        public Cat(int weight)
        {
                this.weight = weight;
        }
        public void setWeight(int w)
        {
                weight = w;
        }
        public int getWeight()
        {
                return weight;
        }
        public void eat()
        {
                System.out.println("Slurp, slurp");
        }
        public  int mealsPerDay()
        {
                return  2 + weight/50;
        }
}
```

```
// here is the inheritance part!

 public class Leopard extends Cat      // "extends" indicates inheritance
{
          protected int numSpots;
          public Leopard()
          {
                    super(); // default constructor of Cat is first called with super()
                     numSpots =0;  // a poor excuse for a leopard!!
          }

          public Leopard(int weight, int numSpots)
          {
                    super(weight);              // a call to the one argument constructor of Cat
                    this.numSpots = numSpots;
          }

          public void setNumSpots(int n)
          {
                    numSpots = n;
          }
          public int getNumSpots()
          {
                    return numSpots;
          }

          public void eat()                 //overriding the eat method of Cat
          {
                    System.out.println("CRUNCH...CHOMP...CRUNCH...SLURP");
          }

          public int mealsPerDay()          //overriding the method of Cat
          {
                    return super.mealsPerDay() * 2;      // note call to parent method
          }
          public void roar()          //a new non-inherited method
          {
                    System.out.println("GRRRRRRRRRRRRRRRRRRR");
          }

}
```

// Use both classes

```
public class CatAndLeopard
{
	public static void main(String[] args)
	{
		Cat felix = new Cat(10);
		Leopard brutus = new Leopard(100,300);
		Leopard lulu= new Leopard();

		felix.eat();
		brutus.eat();
		System.out.println();
		System.out.println("Felix weight: "+felix.getWeight());
		System.out.println("Brutus weight: "+brutus.getWeight());
		System.out.println("Lulu weight: "+lulu.getWeight());
		System.out.println();
		  brutus.roar();
	}
}
```

OUTPUT

```
Slurp, slurp
CRUNCH...CHOMP...CRUNCH...SLURP

Felix weight: 10
Brutus weight: 100
Lulu weight: 10

GRRRRRRRRRRRRRRRRRRR
```

Here is some terminology

- Leopard is a subclass of Cat. Cat is a super class. Leopard is a child class of Cat. Cat is a parent

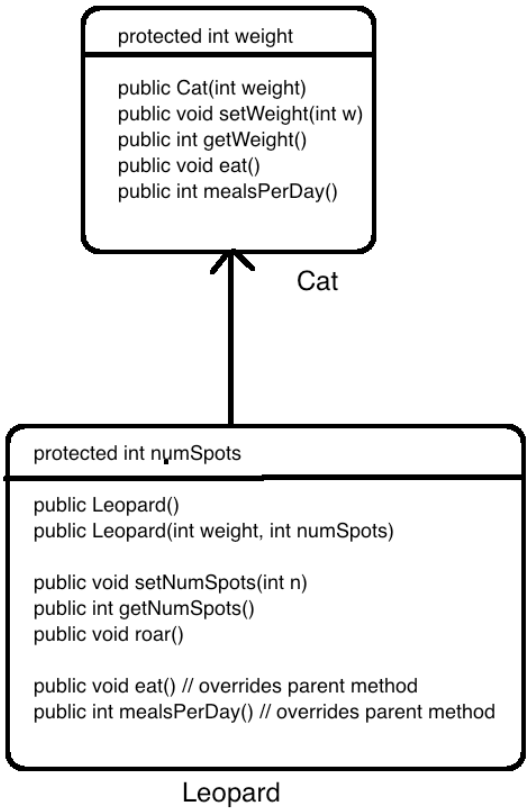
- The keyword *protected* means that a protected varible of a class can be accessed in directly in any of its subclasses. The variable is essentially “inherited” by the subclasses.

- Cat is an ordinary class, no different than any we have seen before except for the keyword *protected*

- ***Leopard extends Cat*** means that the Leopard class gets all the variables and methods of Cat **except the constructors.**

- Leopard is not exactly the same as Cat→ Leopard has more:
  Leopard has the additional variable,numSpots
  Leopard has additional methods; int getNumSpots() and void setNumSpots(int n)

- Leopard overrides the methods mealsPerDay() and eat() inherited from Cat. This means that Leopard has its own versions of these. If Leopard wants to use the Cat versions of these methods. Leopard can call them as super.eat() or super.mealsPerDay()



Leopard extends Cat

Leopard inherits from Cat

Cat is the base Class, Leopard the derived class

Cat is the parent class; Leopard the child

Cat	Leopard	
weight	weight	Leopard inherits from Cat
setWeight()	setWeight()	Leopard inherits from Cat
getWeight()	getWeigtht()	Leopard inherits from Cat
eat()	eat()	Leopard overrides Cat's version
mealsPerDay()	mealsPerDay()	Leopard overrides Cat's version
-----	setNumSpots()	In Leopard only
-----	getNumSpots()	In Leopard only
-----	roar()	In Leopard only

- Leopard does **NOT** inherit constructors and defines its own constructors
- Before a Leopard object is created the Cat constructor must be called.
- So, when creating a Leopard object a Cat object is first made then the Leopard object is created from that.
- Leopard can use super(...) to call a Cat constructor.
- If Leopard does not directly call the Cat constructor then the default Cat constructor is automatically called .

IMPORTANT: CONSTRUCTORS ARE NOT INHERITED

Look at the constructor

```
public Leopard(int weight, int numSpots)
{
        super(weight);
        this,numSpots = numSpots;
}
```

The keyword super calls the one-argument constructor of the parent (base) class. If no call to a constructor of a base class is made, the default constructor of the base class is automatically called:

```
public Leopard() // default constructor
{
        numSpots = 0;
}
```

Here, the default constructor of Cat is first called and numSpots is set to 0.

This can also be written as

```
public Leopard() // default constructor
{
        super(); // calls default of Cat
        numSpots = 0;
}
```

If a call using super(…) is not made and the parent class has no default constructor, then a compilation error will occur. Good practice: Always provide a default constructor for your classes.

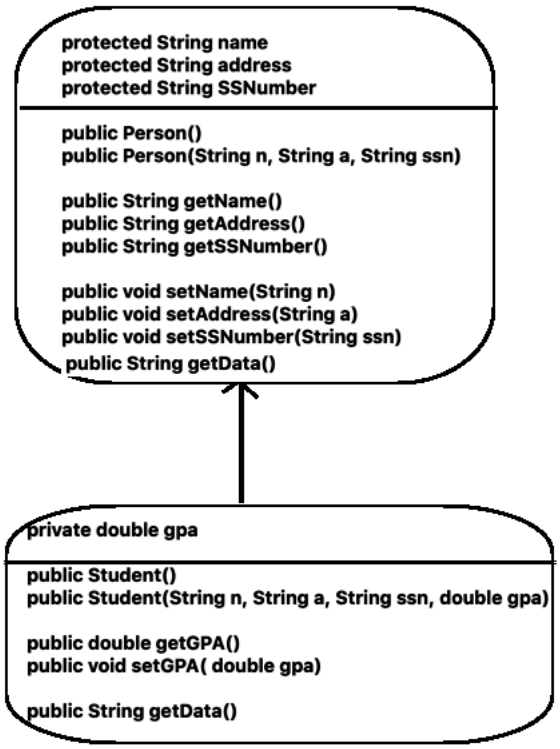
The is-a relationship

The relationship between the parent and the child classes is called an ***is-a relationship*** because every child is-a parent. Notice every Leopard is-a Cat.
If you cannot apply the is-a relationship then inheritance is not appropriate.
Another Example:

```
public class Person
{
  protected String name;
  protected String address;
  protected String SSNumber;
  public Person()
  {
    name = “”; address = “”; ssNumber =””;
  }
  public Person(String n. String a, String ssn)
  {
    name = name; address = a; SSnumber = ssn;
  }
  // getters and setters go here

  public String  getData()
  {
     return name +” “+address+” “+SSNumber;
   }
}
```

```
public class Student extends Person
{
    private double gpa;

    public Student()
    {
       puper();  // call default constr of Person
       gpa = 0.0;
     }
    public Student(String n, String a,
                          String ssn,  String gpa)
    {
       super(n,a,ssn); // call parent constr
       this.gpa = gpa;
    }
    //getters and setters from
    // Person  are inherited
    //  needs getGPA() and setGPA()

  public String getData()
  {
    peturn super.getData()+” “+gpa;
  }
}
```

Inheritance makes sense here because **every student is-a person**



Student inherits the data (name, address, ssNumber) from Person
Student inherits the getter and setter methods from Person
Student overwrites getData() and has its own version of get Data()

Notice:

- Student constructors call the constructors of Person with super(..)
- Student overrides getData() from person.

```
public String getData()
{
  return super.getData() + “  “+ gpa;
}
```

Super.getData() calls the getData() from the parent, which returns
name + address + ssNumber
and this version of getData() appends the gpa to that information

Here is another simple inheritance example with **both private and protected** variables:

```
 public class Parent
{
   private int x;
   // not directly accessible to subclasses
   protected int y;
   // directly asscessible to subclasses

   public Parent() // default Constructor
   {
       x = 1;
       y = 2;
   }
   public int getX()
   {
       return x;
   }
   public void setX(int x)
   {
      this.x = x;
   }
    public int gety()
   {
       return x;
   }

   public void setY(int y)
   {
      this.y = y;
   }
}
```

```
public Child() extends Parent
   {
      super();
      // a call to the default constructor of Parent
   }

   public void printValues()
   {
       System.out.println ( "The value of x is  "+ getX);
       // x is private in  Parent   -- no direct access

       System.out.println ("The value of y is    "+   y);
       // y is protected the Child class has direct access
   }


   public void setValues(int x, int y)
   {
        setX(x);
        // x is private in Parenty, no direct access

          this.y = y;
       // y is protected in Parent,  Child has access
   }
}
```

```
public class ParentChild
{
   public static void main(String[] args)
   {
      Child ch = new Child();
      System.out.println(ch.getX());
      ch.setX(100);
      ch.setY(200);  // need setter outside the hierarchy
      ch.printValues();
   }
}
```

When **NOT** to use inheritance

Here is an example when inheritance is not appropriate. The code will run but logically but the is-a relationship does not apply.

```
public class Point
{
   protected int x,y;
   public Point()
   {
      x = y = 0;
   }
   public Point(int x, int y)
   {
       this.x = x ;
       this.y = y;
   }
   public int getX()
   {
       return x;
   }
  public int getY()
   {
       return y;
   }
   public void setX(int x)
   {
       this.x = x;
   }
  public void setY(int y)
   {
       this.y = y;
   }
}
```

```
public class Circle extends Point
{
  // this is not really appropriate, WHY
  // Circle inherits x and y from Point
  // (x,y) is the center of thcircle

  private int radius;
 public Circle()
 {
     super();
     radius = 1;
 }
 public Circle(int x, int y, int radius)
 {
      super(x,y) ;
      this.radius = radius ;
  }

  public String equation()
       // returns the equation of a circle
      // with center (x,y) and radius, radius
  {
   return "(x –"  +  x  + ")^2)"  +  " + "+
              "(y –" + y  + ")^2)"  + " = "+
               radius*radius;    }
}
// Circle C = new Circle (3,4,10) ;
//System.out.println( c.equation())
// Output is
//  (x –3)^2) + (y –4)^2) = 100
```

This is not a logical use of inheritance
The is-a relationship does not hold
Every Circle IS NOT a POINT
This is a has-a relationship
We can say " Every Circle has-a Point "
(it's center)

This is called Composition
Here is a better design.
A Circle HAS_A point, its center

```
public class Circle1
{
    private Point center;  // Circle has-a Point
    private int radius;

    public Circle1()
   {
     center    = new Point() ;
     radius  = 1 ;
   }
  public  Circle1 (int x, int y, int radius)
  {
     center = new Point(x,y) ;
    this.radius = radius ;
 }
 public String equation()
 {
   return "(x –"  + center.getX()  + ")^2)"  +  " + "+
             "(y –" + center.getY()   + ")^2)"  + " = "+
              radius*radius;
 }

 public static void main(String[] args)
 {
   Circle1 c = new Circle1(3,4,10);
   System.out.println(c.equation());
 }
}
```