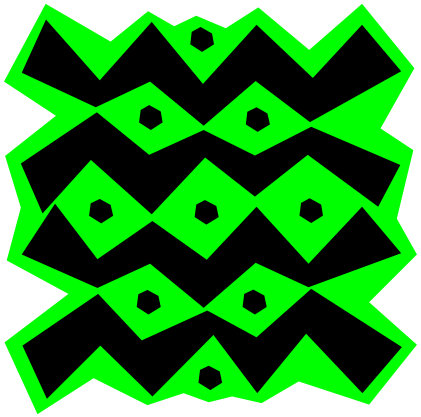
Abstract Classes

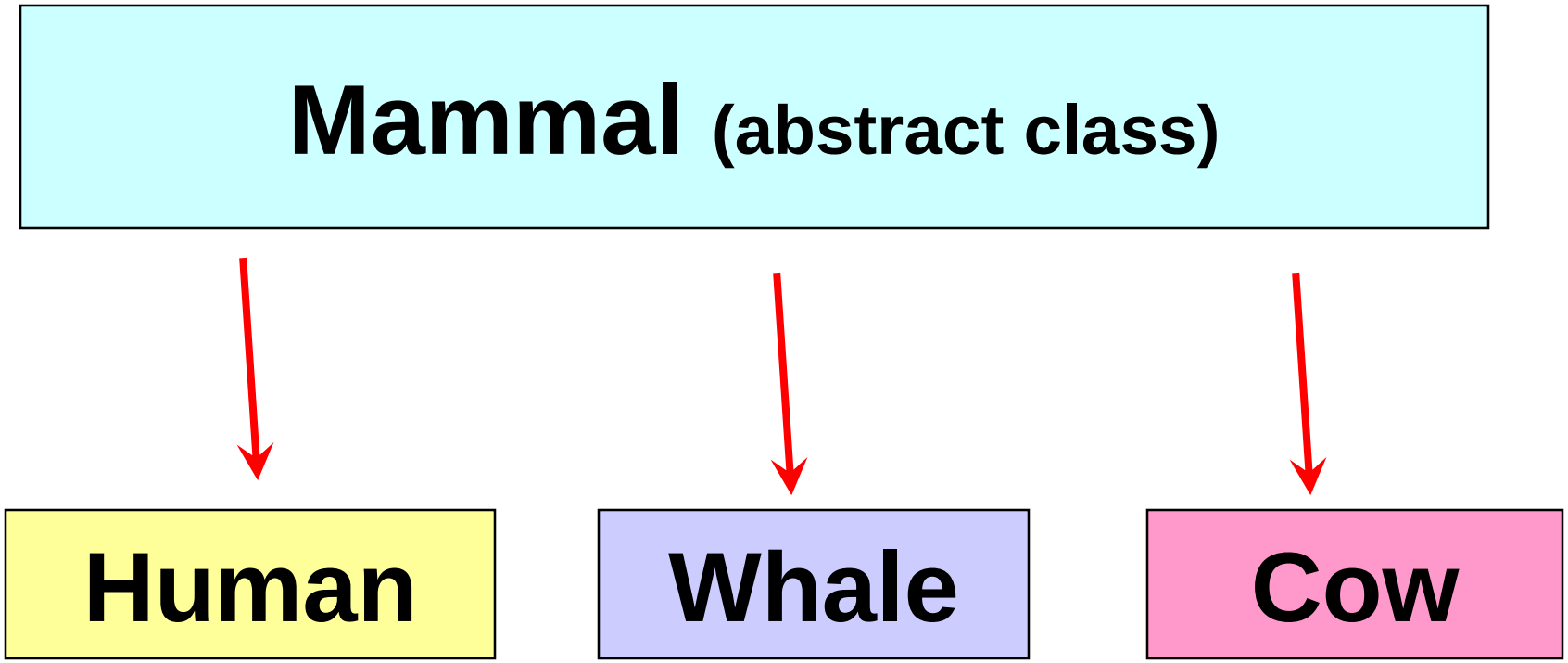


Abstract Classes

Abstract classes are used to define a class that will be used only to build new classes.

No objects will ever be instantiated from an abstract class.

Real Abstract Class



Abstract Classes

Any sub class that extends a super abstract class must implement all methods defined as abstract in the super class unless the extending class is an abstract class.

Abstract Classes

Abstract classes are typically used when you know quite a bit about an Object and what you want the Object to do, but yet there are still a few unknowns.

Abstract Class

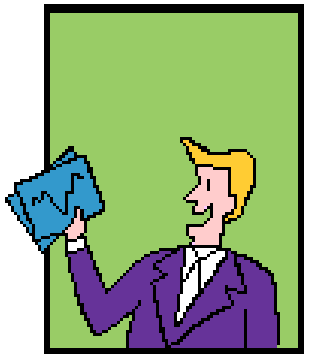
```
public abstract class Monster
{
   private String name;

   public Monster( String nm )
   {
      name = nm;
   }

   public abstract String talk( );

   public String toString()
   {
      return name + " says  "  + talk();
   }
}
```

Abstract Classes

Why define talk as abstract?

```
public abstract String talk( );
```

Does each Monster say the exact same thing?



```
public class Vampire extends Monster
{
  public Vampire( String name )
  {
     super(name);
  }

  public String talk()
  {
     return  "\"I want to drink your blood!\"";
  }
}
```

Vampire

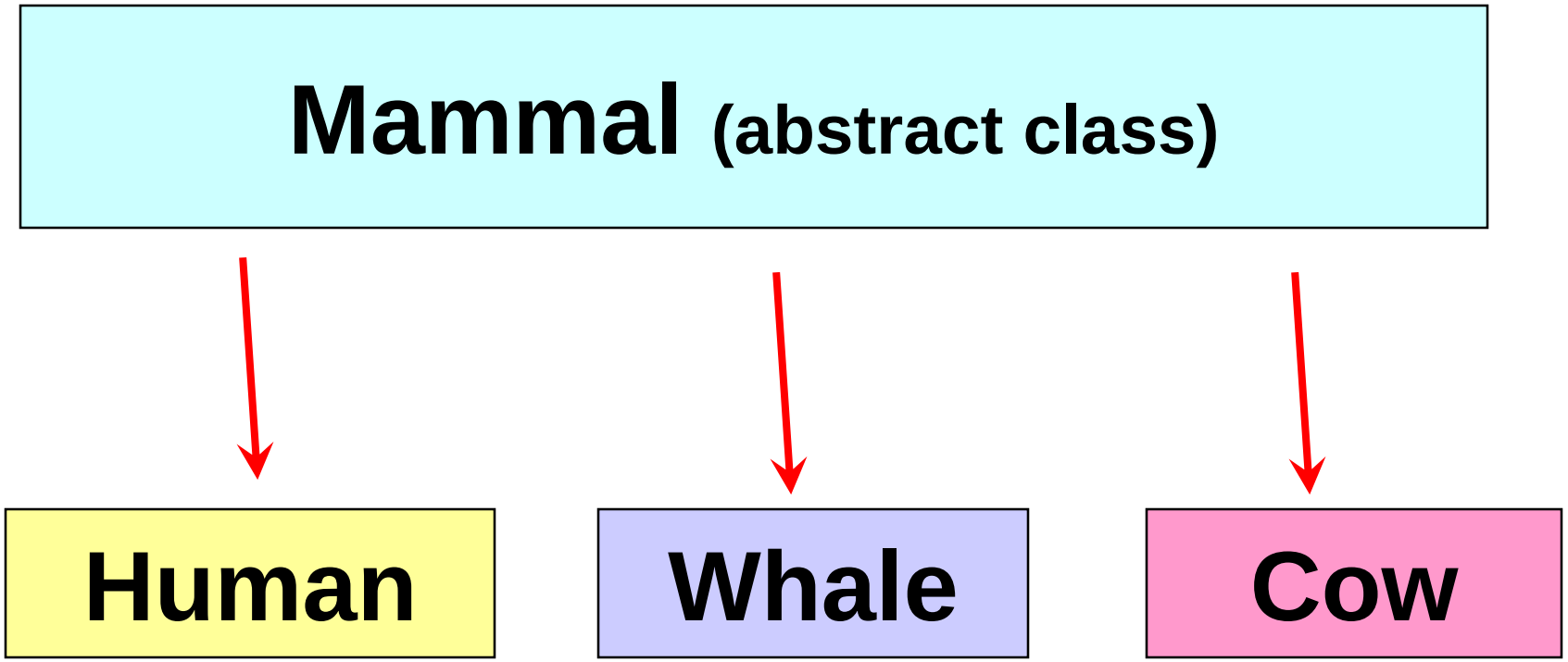
Sub Class

```
public class Ghost extends Monster
{
  public Ghost( String name )
  {
    super(name);
  }

  public String talk()
  {
    return  " \"Where did I go?\"\n\n";
  }
}
```

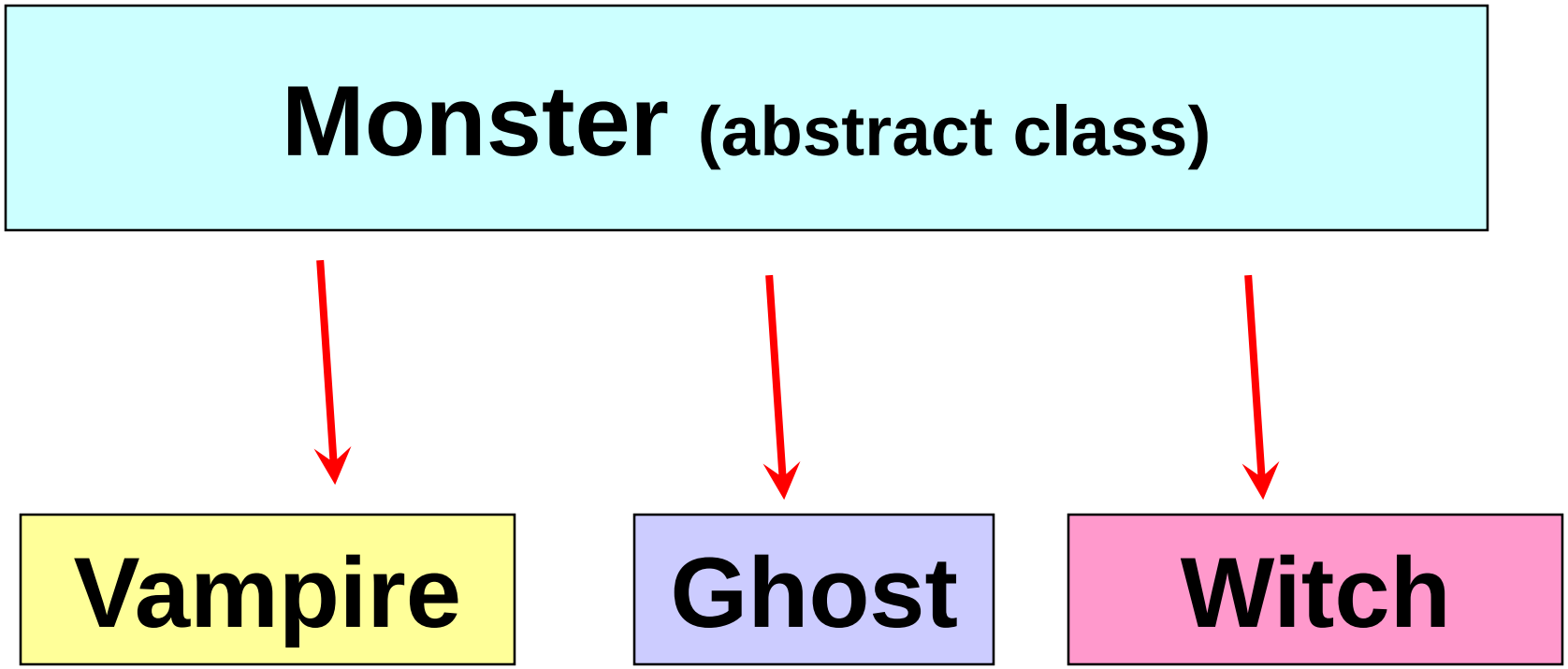
Ghost

Sub Class

Abstract Classes



Abstract Classes



Open monster.java ghost.java ghostrunner.java

Polymorphism

Polymorphism - the ability of one general thing to behave like other specific things.

Polymorphism

```
//instance variable
private Monster[] monsters;

//ask for the number of monsters
//get the number of monsters

for ( int j=0; j < monsters.length; j++ )
{
  out.print("Enter Monster " + j + " Name :: ");
  int r = (int)( Math.random() * 3 );
  if(r==0)
      monsters[j] = new Vampire(kb.nextLine());
  else if(r==1)
      monsters[j] = new Witch(kb.nextLine());
  else
      monsters[j] = new Ghost(kb.nextLine());
}
```

Polymorphism

```
public String monstersTalk( )
{
   String out = "";
   for ( int i=0; i<monsters.length; i++ )
      out += monsters[i].talk();
   return out;
}
```

Polymorphism

```
public String toString( )
{
   String output="";
   for ( int i=0; i<monsters.length; i++ )
      output+=monsters[i].toString();
   return output;
}
```

Open monsterpack.java packrunner.java

Binding

```
class Human { .... }

class Boy extends Human{

   public static void main( String args[]) {

      /*This statement simply creates an object of class *Boy
      and assigns a reference of Boy to it*/
      Boy obj1 = new Boy();

      /* Since Boy extends Human class. The object creation *
      can be done in this way. Parent class reference * can
      point to a child class object*/
      Human obj2 = new Boy();
     //Boy obj2 = new Human(); Never do this!
     Boy is a type of Human – Human IS NOT a type of Boy
   }

}
```

Static Binding

Method calls are locked down at compile time based on the type of reference used.

```
Object o = new String("dog");
int len = o.length();     //syntax error
                          //object has no length
int len = ((String)o).length();   //add a cast
```

Static Binding

Method calls are locked down at compile time based on the type of reference used.

```
Actor a = new Bug(Color.GREEN);
a.move();                          //syntax error
                                   //Actor has no move

((Bug)a).move();                   //add a cast
```

Open staticbinding.java

Dynamic Binding

Specific types of objects associated with method calls are determined at run time, creating polymorphic behavior.

```
public void monstersTalk( )
{
   out.print("monstersTalk\n\n");
   for ( int i=0; i<monsters.length; i++ )
      out.println( monsters[i].talk() );
}
```

Dynamic Binding

```
public double processList( List<Integer> list )
{
   double sum = 0;
   for( int i = 0; i < list.size(); i++ )
      sum += list.get(i);
   return sum / list.size();
}
```

Calls to processList() could be made with an ArrayList, LinkedList, Vector, or Stack as all four classes implement the List interface, sharing a common set of methods.

Open dynamicbinding.java

Description	Interface	Abstract Class
Can contain abstract methods?	Yes	Yes
Can contain non-abstract methods?	No	Yes
Can contain constructors?	No	Yes
Can be instantiated?	No	No

Description	Interface	Abstract Class
Can be extended?	Yes	Yes
Can be implemented?	Yes	No

Description	Interface	Abstract Class
Can contain instance variables?	No	Yes
Can contain final instance variables?	No	Yes
Can contain final class variables?	Yes	Yes
Can contain class variables?	No	Yes

Extends / Implements RULES

Classes extend Classes

Interfaces extend Interfaces

SAME extends SAME

Classes implement Interfaces

CLASS implements INTERFACE

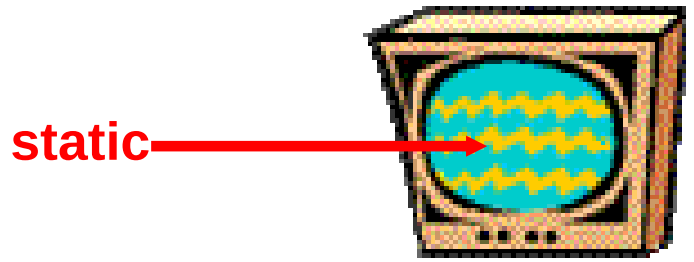
Open classextends.java interfaceextends.java

What is static?

static

Static is a reserved word use to designate something that exists as part of a class, but not part of a specific object.

Static variables and methods exist even if no object of that class has been instantiated.

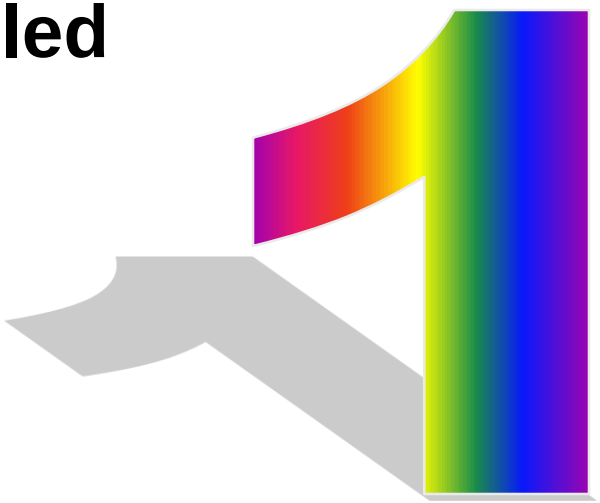


static

Static means one!

All Objects will share the same static variables and methods.

Static variables are also called class variables.



static

```
class Monster
{
   private String myName;
   private static int count = 0;

   public Monster() {
     myName ="";
     count++;
   }
   public Monster( String name ) {
     myName = name;
     count++;
   }
}
```

all Monster share count

Open static.java

Start work on the Labs