Interfaces and Abstract Classes

- What are they?
- How are they used?
- Where will we use them?
- Calling superclass' constructors and methods

Abstract Class – A class which at least one of the methods is left "abstract", or without the code implemented.

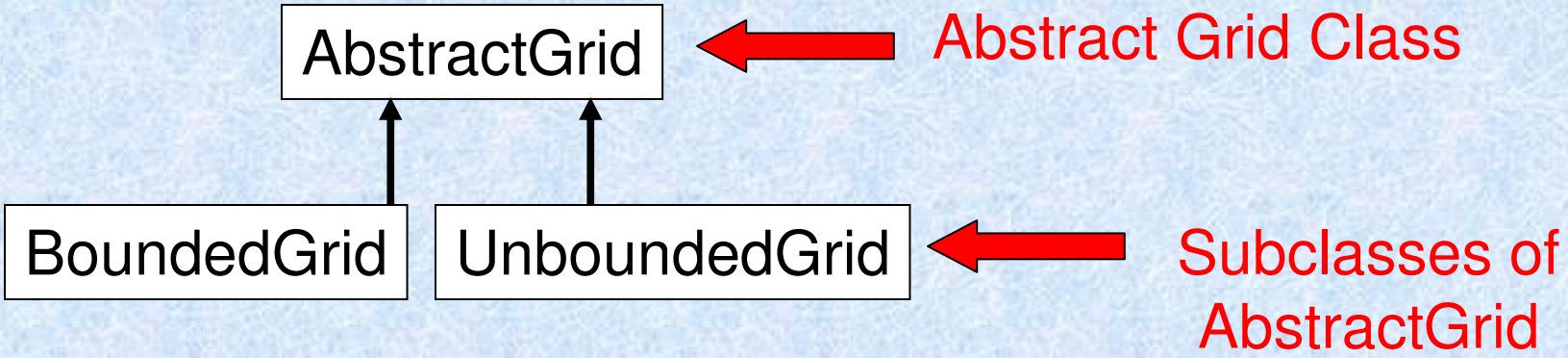
```
public abstract class Polygon
{
        private int numberOfSides;

        public Polygon()
        {       /* code not shown */   }

        public abstract double getArea();

        /* other methods not shown */
}
```

Abstract Class – A class which at least one of the methods is left "abstract", or without the code implemented.

```
public abstract class Polygon
{
        private int numberOfSides;

        publ
        {

        publ

        /* c
}
```

```
public class Rectangle extends Polygon
{
        private double myLength;
        private double myWidth;

        public Rectangle()
        {       /* code not shown */   }

        public double getArea()
        {
                return myLength * myWidth;
        }

        /* other methods not shown */
}
```

Example: Inheritance Diagram for GridWorld's Grid



Interface – A construct in Java where everything is left "abstract". Interfaces have no constructors, instance variables, nor program code.

```
public interface Fillable
{
        void fill(int x);
        int getCurrentAmount();
        int getMaximumCapacity();
}
```

Interface – A construct in Java where everything is left "abstract". Interfaces have no constructors, instance variables, nor program code.

```
public interface Fillable
{
        voic
        int
        int
}
```

```
public class Car implements Fillable
{
        ...
        public void fill(int gallons)
        {       fuelAmount += gallons; }

        public int getCurrentAmount()
        {       return fuelAmount;     }

        public int getMaximumCapacity()
        {       return fuelTankCapacity;      }
}
```

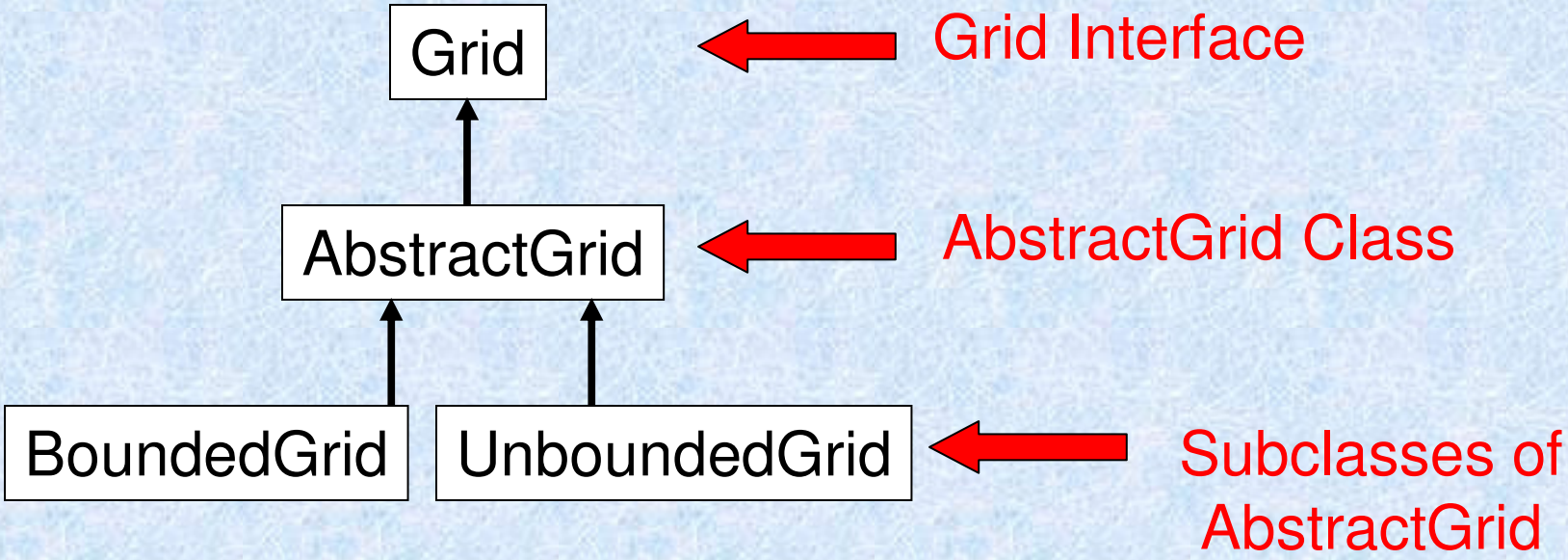
Interface – A construct in Java where everything is left "abstract". Interfaces have no constructors, instance variables, nor program code.

```
public interface Fillable
{
        voic
        int
        int
}
```

```
public class Car implements Fillable
{
        ...
}
```

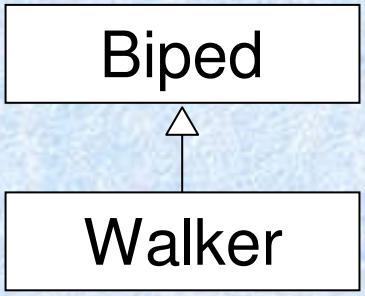
```
public class VendingMachine implements Fillable
{
        ...
        public void fill(int quantity)
        {       currentStock += quantity;     }

        public int getCurrentAmount()
        {       return currentStock;   }

        public int getMaximumCapacity()
        {       return 20;     }
}
```

Example: An Interface used in GridWorld Simulation



A class that implements an Interface MUST implement ALL of the methods defined in the Interface.

Calling Superclass's Constructors




```
public class Walker extends Biped
{
    // Constructor
    public Walker(int x, int y, Image leftPic, Image rightPic)
    {
        super(x, y, leftPic, rightPic);
        ...
    }
}
```

Calls **Biped**'s constructor

The number / types of parameters passed to **super** must match parameters of one of the superclass's constructors.

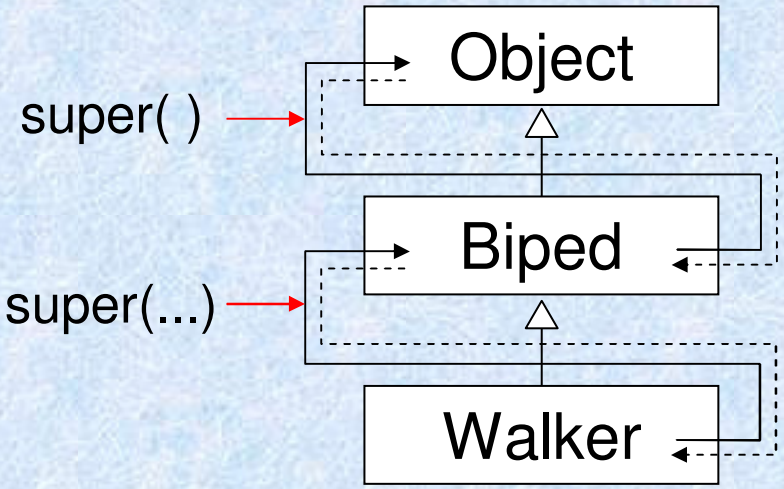
If present, must be the first statement

Calling Superclass's Constructors (cont'd)

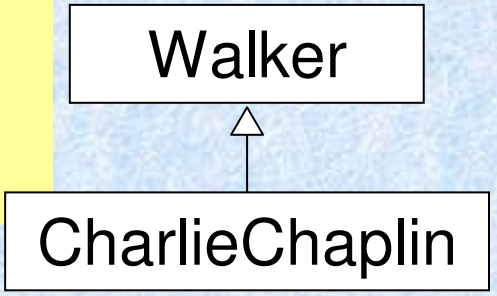
- One of the superclass's constructors is always called, but you don't have to have an explicit super statement.

Calling Superclass's Constructors (cont'd)

- Superclass's constructor calls its superclass's constructor, and so on, all the way up to Object's constructor.



Calling Superclass's Methods




```
public class  CharlieChaplin
                        extends Walker
{
    ...
    public void nextStep ()
    {
        turnFeetIn();
        super.nextStep();
        turnFeetOut();
    }
        ...
}
```

Calls **Walker**'s **nextStep**

super.*someMethod* refers to ***someMethod*** in the nearest class, up the inheritance line, where ***someMethod*** is defined.