Casting --> temporarily changes the type

Two Types: implicit (implied) and explicit (stated)

Rule of Thumb: implicit casting can occur, if precision is not lost

float pi = 3.1415926;

int i = pi;  // requires an explicit cast

int i = (int)pi;

What about?

int i = (int)(pi/2);
// yes, this will work. (pi/2) is an implicit cast; 2 becomes 2.00...
Balloon myBalloon = new Balloon("green", 10,10);

// myBalloon can be recast as an object, because it is an object
aMethod((Object)myBalloon)

public void aMethod(Object o)
{
    //do something
}
Balloon myBalloon2 = new Balloon("yellow", 20,30);
String str = "a string";

Object[ ] obj = {(Object)myBalloon2, (Object)str};
anotherMethod(obj);

private void anotherMethod(Object[ ] o)
{
    :
    // here we use two concepts: selecting array elements and casting
    String s = (String)o[1];
    Balloon b = (Balloon)o[0];
}
**Scope** specifies the range of influence of a variable, method, or parameter

The scope of a method is the entire class

The scope of a variable depends on where it is located: variables not contained within a method are global to the entire class, whereas variables contained within a method are local to that method

```java
public class MyClass {
    private int i,j;  //i and j are global variables
    private void aMethod() {
        int k = i*j;  //k is a local variable
    }
}
```
public class MyClass
{
    private static int i, j;  // I and j are global variables

    public static void main(String[] args)
    {
        i = 3;
        j = 2;
        aMethod();
    }

    private static void aMethod()
    {
        int k = i * j;  // k is a local variable
        System.out.println(k);
    }

    private static void anotherMethod()
    {
        System.out.println(k);  // "k" is undefined in anotherMethod
    }
}
Can an access modifier broaden or narrow the scope? Yes.

```java
public class MyClass {
    private static int i, j, k; // I and j are global variables

    private static void aMethod() {
        k = i * j; // k is now global
        System.out.println(k);
    }
}
```
public class MyClass
{
    private static int i,j,k;  // i and j are global variables

    public static void main(String[] args)
    {
        i = 3;
        j = 2;
        aMethod();
    }

    private static void aMethod()
    {
        k = i*j;  // k is a global variable
        System.out.println(k);
    }

    private static void anotherMethod()
    {
        System.out.println(k);
    }
}
public class MyClass
{
    private static int i, j, k;  // I and j are global variables

    public static void main(String[] args)
    {
        i = 3;
        j = 2;
        aMethod(i, j);
    }

    // parameters a and b are local to aMethod
    private static void aMethod(int a, int b)
    {
        k = a * b;  // k is a local variable
        System.out.println(k);
    }

    private static void anotherMethod()
    {
        System.out.println(k);
    }
}
Access Specifier: private, public, protected, and friendly (default)

private – scope restricted to current class in which it’s contained

public – scope open to all classes

protected – scope available within the current class, to subclasses, or classes not even in the same package, but not to classes outside of the member’s inheritance tree.

friendly – “package” scope open to all classes in the current package
Variable types:
http://journals.ecs.soton.ac.uk/java/tutorial/java/nutsandbolts-vars.html

Java keywords:
http://java.sun.com/docs/glossary.html