Inheritance - Classes & Children: **abstract**

Avoid ambiguities - Java allows extending only 1 class

**Interface**: like an abstract class with no data, just specify some methods that are allowed

```java
public interface MyInterface {
    // contract!
    public void print();
}
```

```java
public class MyClass extends MyParent implements MyInterface {
    // Data
    // Methods
    public void print() {
        // code appropriate to print this object
    }
}
```
Example: Circular class

Comparable interface → compareTo method

01. compareTo (o2) -1 if o1 smaller
0 if equal
+1 if o1 bigger

public abstract class Circular implements Comparable<Circular>

// Circular objects among themselves

// radius
// constant Pi

// constructor...

public abstract double getArea ()

public int compareTo (Circular c)

if (this . getArea () < c . getArea ()) return -1;
if (getArea () > c . getArea ()) return +1;
return 0;

// available in Circle, Sphere

// rewrite in circle class if we want it different or
more specialized.
```java
Circle c = new Circle (1);
Sphere s = new Sphere (2);
System.out.println (c.compareTo(s));
BankAccount b = new BankAccount (100);
System.out.println (c.compareTo(b));
```

Compilation error

// Test Circle file using packages
import CircularObjects;
I want to use all classes defined in this package
import java.io; // library (pre-defined classes & interfaces)

Read the java documentation!

Classes in a package have "privileged" access to each other public/private; no keyword -> visible inside package not outside.
Comparable, Serializable

double[] a;
content not address
Save in such a way that we can re-create object
Public class MyClass implements Serializable {

Look up in Java doc what methods
need to be specified
I can implement other interfaces Comparable,
MyInterface...

Organizing code & packages

package CircularObjects;
public abstract class Circular {

3 // Circular.java

package CircularObjects;
public class Circle extends Circular {

3 // Circle.java
Sorting class:

```java
Comparable <T> [] a;

a[i].compareTo(a[j]);
```

Java figures out what method to call here based on type of objects in a

Exceptions - errors

```java
public void MyMethod() throws Exception {
}
```

Either pass it to caller or fix problem

```java
try {
    MyMethod();
} catch (Exception e) {
    // code to deal with the exception
}
```

Exceptions are classes like all other Java constructs.

```java
public class MyException extends Exception {
    customized with messages
}
```
Comparing objects to determine if they are the same

Objects in Java are references:

```java
Circle c, d;
c = new Circle(1);
d = new Circle(1);
if (c == d) {
    // compares references, which are not same.
    c = d;
}
```

Method equals compares content, return true/false
if (c.equals(d))

```java
// In Circle class
public boolean equals(Circle d) {
    return radius == d.radius;
}
```