

CHAPTER -8

Interfaces and Abstract Classes(EXERCISE)

① equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();

if (a.equals(c)) {
    System.out.println("true");
} else {
    System.out.println("false");
}
```

```
File Edit Window Help Stop
% java TestObject
false
```

Tells you if two objects are considered 'equal'.

③ hashCode()

```
Cat c = new Cat();
System.out.println(c.hashCode());
```

```
File Edit Window Help Drop
% java TestObject
8202111
```

Prints out a hashCode for the object (for now think of it as a unique ID).

② getClass()

```
Cat c = new Cat();
System.out.println(c.getClass());
```

```
File Edit Window Help Faet
% java TestObject
class Cat
```

Gives you back the class that object was instantiated from.

④ toString()

```
Cat c = new Cat();
System.out.println(c.toString());
```

```
File Edit Window Help LapseintoComa
% java TestObject
Cat@7d277f
```

Prints out a String message with the name of the class and some other number we rarely care about.

Everything comes out of an `ArrayList<Object>` as a reference of type `Object`, regardless of what the actual object is or what the reference type was when you added the object to the list.

The objects go IN as **SoccerBall**, **Fish**, **Guitar**, and **Car**.



But they come OUT as though they were of type **Object**.



Objects come out of an `ArrayList<Object>` acting like they're generic instances of class `Object`. The Compiler cannot assume the object that comes out is of any type other than `Object`.

you are here ▶ 213

EXERCISE

2.What's the Declaration?

2.

```
public abstract class Top { }
public class Tip extends Top { }
```
3.

```
public abstract class Fee { }
public abstract class Fi extends Fee { }
```
4.

```
public interface Foo { }
public class Bar implements Foo { }
public class Baz extends Bar { }
```
5.

```
public interface Zeta { }
public class Alpha implements Zeta { }
public interface Beta { }
public class Delta extends Alpha implements Beta { }
```

3.POOL PUZZLE

```
interface Nose {
public int iMethod() ;
}
abstract class Picasso implements Nose {
    public int iMethod() {
        return 7; }
}
class Clowns extends Picasso { }
class Acts extends Picasso {
    public int iMethod() {
        return 5; }
}
public class Of76 extends Clowns {
    public static void main(String[] args) {
        Nose[] i = new Nose [3] ;
        i[0] = new Acts() ;
        i[1] = new Clowns() ;
        i[2] = new Of76() ;
        for (int x = 0; x < 3; x++) {
            System.out.println(i[x].iMethod() + " " + i[x].getClass());
        }
    }
}
```

OUTPUT

```
5 class Acts
7 class Clowns
7 class Of76
```