What is common to a Cat and a Person ?
Breath, Move, Eating

```
CanHavePizza eater = new Cat();
Animal eater = new Cat();
Interface CanHavePizza {
            int PIZZASIZE;
             void eatPizza();
}
```

They both can have pizza!

```
Class Restaurant {
            boolean servePizza(CanHavePizza eater){
                        eater.eatPizza();
        if(eater instance of Person)
                                    // process payments
}
}
delpapa.servePizza(new Cat());

Class Student extends Person implements CanHavePizza, CanHaveRetake, CanHaveParty,
Movable
Interface CanHaveParty{
        move
  dance
}
Interface Movable {
        move
}
Class Cat extends Animal implements CanHavePizza
```

CLASSES
Inheritance isA
WHAT/ WHO , methods (behavior) and fields (state)

INTERFACES - very very very abstract class
A is capable of B, Interfaces are used to implement COMMON behavior among DIFFERENT UNRELATED classes
ABSTRACT methods only, no FIELDs (except of final public static), multiple implementation , multiple inheritance
Separation, Reusability, Extensibility, Scalability, Flexibility, Maintab…

1. Interfaces have no constructors, no concrete methods, no objects, no instance fields
2. All methods are public and abstract
3. All fields are static and final

Loosely coupled

```
Interface Pluggable {}
Class PowerSocket{
        boolean charge(Pluggable p){

}
```

PowerSocket ‘talks’ with pluggable

```
Interface Game {
        a
        b
        c
}
Interface IGame extends Game {
        d
}
Class MemoryGame  implements IGame {}
Class LogicGame implements Game{}
Class App {
        void getStatistics (Game g){}
}

Sellable
Pluggable
Interface SellableAndPluggable extends Sellable, Pluggable{}



Class iPhone extends Device implements SellableAndPluggable {}
```

Circle, Rectangle, Triangle, Image, Text, Drawing, NNCreature, BrandNewItem

```
Class Painter{
        Vector<Paintable> objects;
        void showAll(){
                for(Paintable cur: objects) {
                        cur.paint();
}
}
}
Interface Paintable {
        void paint();
        boolean remove();
}
```

Class Shape

Class Text implements Paintable
Class Circle extends Shape implements Paintable
…

2 ways to work with Interfaces: use existing, build your own

Comparable
toString printing
Equals ==
 int compareTo
> 1
< -1
= 0

```
if(a > b) …
if(a.compareTo(b) > 0) ….

For
        for
    if(a[I].gpa>a[j].gpa) swap
```

Counting sort
a 1 2 3 4 1 1 4 5 2 1

b[a[i]]++

```
b 0 1 2 3 4 5 6 7
  0 4 2 1 2 1 0 0
```

Print b[i] times i
1 1 1 1 2 2 3 4 4 5

```
Collections.sort(list, new NameComparator());
Collections.sort(list);

Class NameComparator extends Comparator {
        int compare(Student a, Student b) {
                        return a.name.compareTo(b.name);
        }
}
```