Agenda :

1. Genorics
2. Raw Types
3. Static in Genenics
4. Inheritence in Generics.

Generics :

$$\begin{tikzcd}
\text{List} < \underline{\quad} > \arrow[r, bend left] & \text{datatype.}
\end{tikzcd}$$

List < ___ > friends ;

```
Class Friend {
    String name1;
    String name 2;
    Integer relation;
}
List <Friend> friends ;
```

✓ [["P1", "P2", 10]

✓ ["P2", "P3", 2]

✗ ["P4", "P5", "CLOSE"]

]

- "CLOSE"
- "ENEMIES"
- "FRENEMIES"

```
class Friend2 {
    String name1;
    String name 2;
    String relation;
}
```

["P5", "P6", "0.95"]

0 - 1

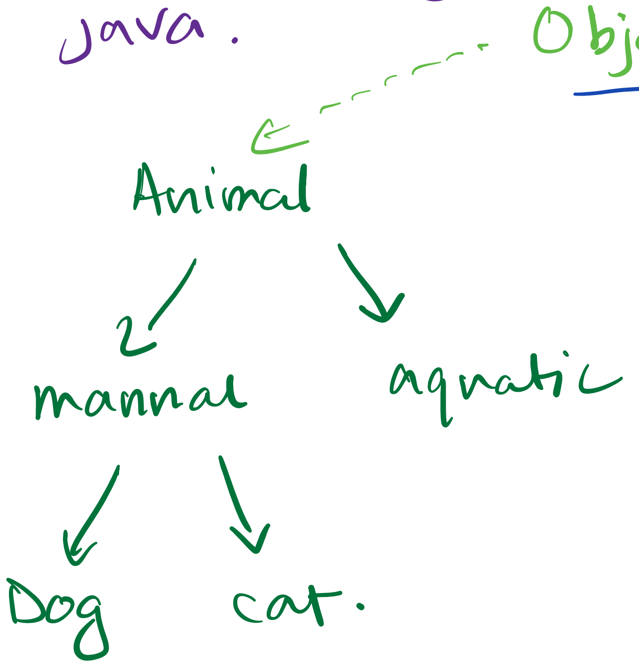
["P6", "P7", false]

```
Class Friend 3 {
    String name1;
    String name 2;
    boolean relation;
}
```

First Solution :

Object Class - It's super/parent class to every class that exists in java.

Object → topmost parent class



```
class Animal extends Object
{
```

↓ this is not written in code, takes place behind the scene.

?

① Animal a = new Dog();
Mammal m = new Dog();
Animal a = new Mammal();

② Object o = new Dog();
Animal();
✓ Integer();
✓ String();
Boolean();

Object reference can point to any object of child class.

```
Class Friend {
    String name1;
    String name2;
    Object relation;
}
```

✓ Friend f1 = new Friend("Akash", "Sam", 2);

✓ Friend f2 = new Friend("Sam", "Tejas", 3);

Friend f3 = new Friend("Sam", "Happy", f1);

↳ No error, but makes no sense.

Object r1 = f1.relation;

Object r2 = f2.relation;

Integer sum = (Integer) r1 + (Integer) r2;

Object r3 = f3.relation ;

Integer sum2 = (Integer) r3 + - - ;

$\swarrow$

RuntimeException

ClassCastException

Problem of Object:

```
Friend f = new Fried ("P1", "P2", "CLOSE");

    f.relation = new Arraylist();
```

Generics.

A concept which allows us to create a class with parameterised data type of their attributes.

```
Class Friend<T>{
}
```

Friend<Integer>

String name1;

String name2;

T relation;

}

f = new Friend();

E, V, T, S, any single digit uppercase letter. [EV, VT, TS]

```
Friend<String> f =
        new Friend("Akan", "Tejas", "CLOSE");

f.relation = 1; X

String r = f.relation;
```

↳ no type casting required.

```
class Friend<T> {                student

    String   name1;   }  ->  roll no
    String   name2;

    T   relation;

}
                        [123, 241, "CLOSE"]
                        ["Akesh", "Sam", "CLOSE"]
                        [123, 12, true]


Class  Friend <T, V> {
    V  student1;            ①                      ②
    V  student2;            Integer Student        String Student
    T  relation;            String relation        Integer
                                                     relation
}

① Friend <String, Integer>  f =  new . . . .
② Friend <Integer, String>  f1 =  . . . .
```

Break till 8:10

Raw Types : $\boxed{< \quad >}$

List < Integer > list = new - - .

HashMap < Integer, String > map = - - - .

key , value .

List < Friend > list2 = -- - .

generic type .

not present

① HashMap _ map = new HashMap( ) ;

② List _ list = new ArrayList ( ) ;

↳ Backward Compatibility.

Map < Object , Object >

list < Object >

Error prone

Static in Generics :

```
class Friend <T>  {
        T  relation ;
   void  printRelation ( ) {
        sout ( relation ) ;
   }

  static void  printNewRelation ( ) {
        sout ( relation )
  }


  static <T> T getRelation ( T var ) {


}
```

```
Friend. getRelation ( new Integer(1) ) ;
Friend < Integer >  f  =  new  - - -
```

String .

```
class Friend <T, V> {
    V stud;
    T relation;

    static <E> void printSomething
                          (E somevar){
        sout(SomeVar).
    }

}
    Integer x = 10;
    Friend.printSomething(x);


static <E> E   returnSome(
                  E SomeVar){
    return someVar;
}
```

$$\begin{tikzcd}
 & & \text{Animal} \arrow[dl] \arrow[dr] & \\
 & \text{Mammal} \arrow[dl] \arrow[d] & & \text{aquatic} \arrow[d] \\
\text{dog} & \text{cat} & & \text{fish}
\end{tikzcd}$$

All this classes are present

```
class Zoo <T extends Mammal>
    T animal;
}

Zoo<Animal> z = new Zoo();
Zoo<Dog> x = new Zoo();
       ↑ is parent
Zoo<Mammal> m = new Zoo();
```

Home work :

- Inheritence in generics - extends
- <?> - wild card in generics.
- Subtyping