next 4 classes

- Generics
- lambda's & Streams
- Collections
- Exception Handling

Agenda :

1. Generic Types
2. Raw types
3. Generic Methods
4. Inheritence in Generics (Bounds)
5. Type Erasure .

Generics :

Store a pair of something :

1. Coordinates (int, int) (double, double)
2. Student name & ID (String, long)
3. city, country pair. (String, String).

V1

```
class Coordinates {
    double x;
    double y;
}
```

```
class StudentPair

class CountryPair.
```

V2

```
class Pair {
    String/double first;
    String second;
}
```

```
Pair p = new Pair(0.5, 0.6);
```

Object

Integer String Double

V3

```
class Pair {
    Object first;
    Object second;
}
```

```
Object o = new Integer();
           new String();
           new Double();
```

```
Pair p = new ("Akash", 3);      fun(p);
Pair p1 = new ("Akash", "Amir"); fun(p1);
```

```
fun ( Pair p ) {
Integer i = (Integer) p.second;
}
```

Too flexible!

Generics → A parameterised data-type.

```
fun1( ) {
  int radius = 45;
return 2 x 3.14 x radius;
}
```

```
fun1( int radius){
return 2 x 3.14 x radius;
}
fun1(30);
fun1(40);
```

```
class Pair<V, S> {
    S first;
    V second;
}
```

order is important.

```
Pair<Integer, Double> p = new Pair<Integer, Double>();
```

↓

```
Pair<Integer, Double> p = new Pair<>();
```

```
class Some<V> {
    V x;
    V y;
}
Some<Integer> s = new Some<>();
```

```
class Some<T> {
    T first;
    T second;
}
```

```
Pair<Integer, String> p = new Pair<>
                          (5, "Akash");

String a = p.second;

Pair<String, String> p1 = new Pair<>
                          ("sam", "Akash");

String x = p1.first;
String y = p.first; → Error
```

compile-time type safety

Raw Types :

- Before Java 5, there was no concept of generics

```
HashMap<Integer, String> hm;
HashMap<String, Double> hm1;

HashMap hm = new HashMap();
```

Inheritence & Generics:

Animal a = new Dog();

Animal
↑
Dog

void printAnimalNames(List<Animal> animals)
{
}

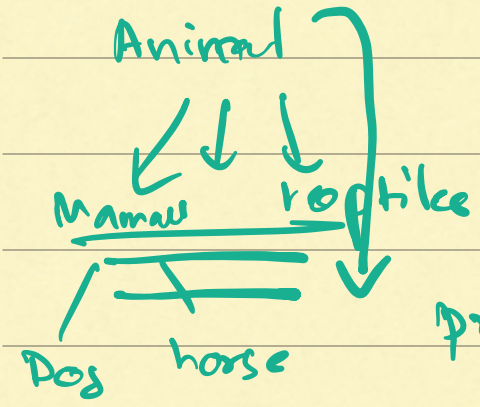
List<Dog> dogs = new ArrayList<>();
printAnimalNames(dogs);

List<Animal>
✗
List<Dog>

animals.add(new Car());
✗

(List<T extends Animal> animals.)

```
void printAnimalNames(List<T extends Animal> animals){
    animals.add(_____);  X
    animals.add(new Dog());  X
}
```



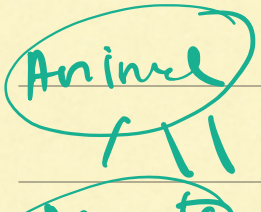

```
List<Mammals> m = new Mammals();
List<Dog> dogs = new Dogs();
printAnimalNames(m);
//          (dogs);
printAnimalNames('cats');
```

upper bound:

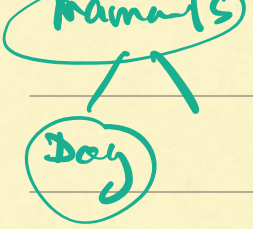
```
void printAnimalNames(List<T extends Animal> animals){
    T a some;
    for(Animal a : animals){
        print(a.name);
    }
}
```

lower bound: (all parent class allowed)

```
void printAnimalNames(List<T super Dog> animals){
```




```
printAnimal names(dogs);
printAnimal names(mammals);
```



printAnimalNames(animals);

WildCard (?)

```
void printAnimalNames(List<? super Animal> animals){
    {
    just printing
    }
```

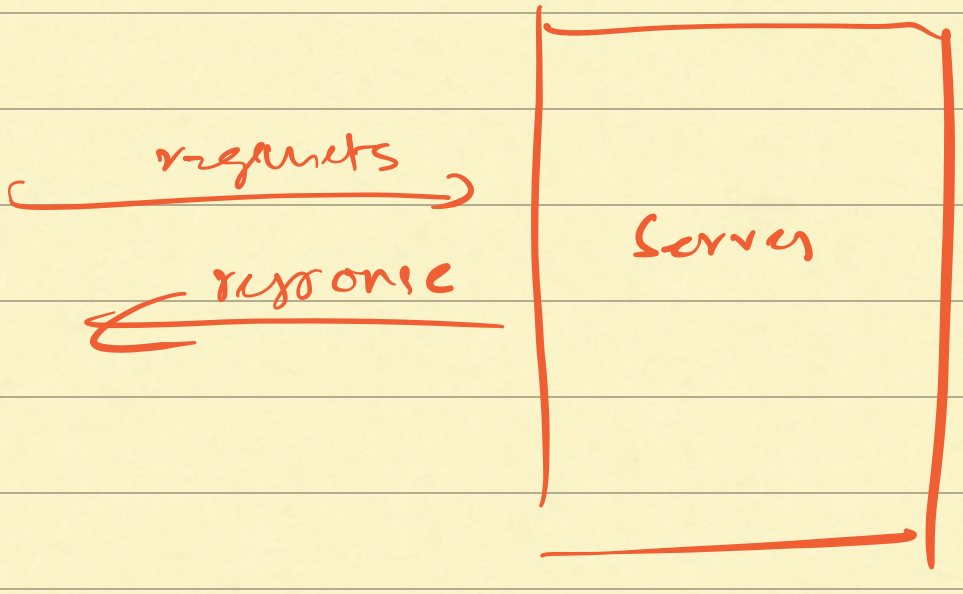
```
void print(List<?> animals){
}
```

Type Erasure :

```
Class Pair <V, S>{
    V first;
    V second;
}
```

At run time both first & second are objects.

```
HashMap hm = new HashMap();
```




```
Student {
    String name;
    float psp;
}
```

```
request.createStudent(url, Student.class)
```

JSON {

name : "Akash"

psp : "1.2"

}

New Object Student

name = "Akash"

psp = 1.2f

ClassPair <V, S> {

V first;

S second;

}

req.createPair (url, Pair.class);

JSON:

{

first : "5.4",

second : "7.2";

}