- Generics
- Collections
- Lambdas & Streams
- Exception Handling

- generic Types
- Raw Types
- Generic methods
- Bounds
- Type Erasure -

List < > coordinates = lat, long

data-type ?

Point

```
class Point {
    double lat;
    double long;
}
```

```
class StudentPerfMetrics {
    double psp;
    double attendance;
}
```

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

Object

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

→ too flexible

Java 5

generics

```
Pair x = new Pair();
x = "Mohit";
y = 5;
```

Collection ⟶ Stack

container

```
Stack {
    ?[] array;
Object[] array;
}
```

parameter

argument

```
int fun(int x){
    x += 5;
    return x;
}
```

fun(5)

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

T, V, E

```
Pair<Integer> pair = new Pair<>();
```

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

```
Pair<Integer, String> pair = ____;
```

parameterised Types

<E>

<T>

very open

Object

```
<E> void fun(List<E> list){

}
```

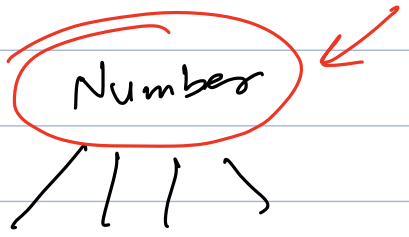
List < E extends Number>

List<Integer>

List<Double>

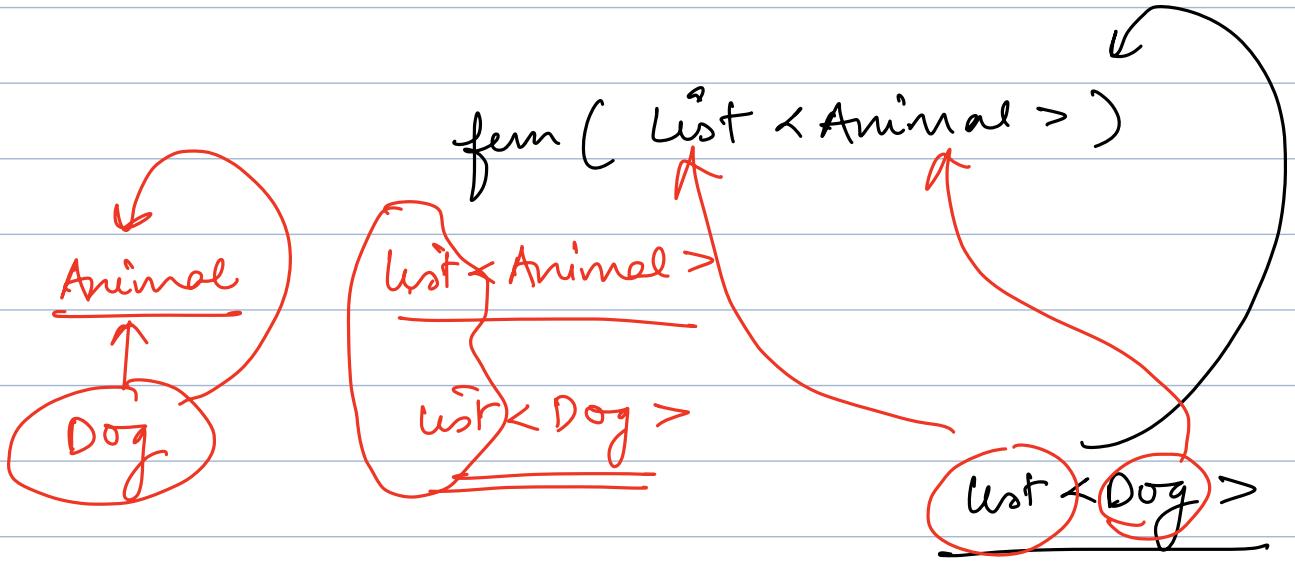
;

List<Animal>




```
Dog d = new Dog();
Animal x = new Dog();

List<Animal> x = new List<Dog>
```




```
List<Animal> a = new ArrayList<>();
a.add(new Dog());
```

Type Erasure

↓

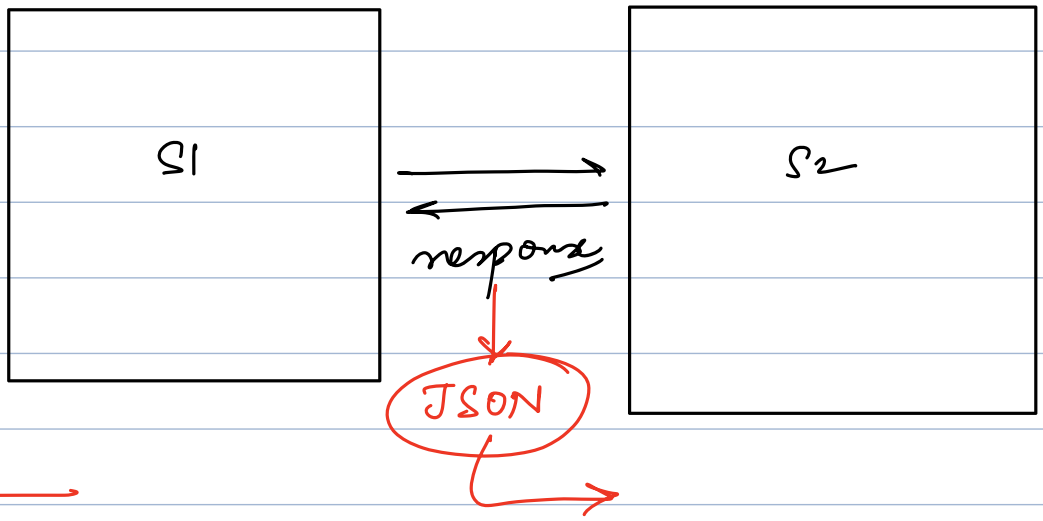
erase

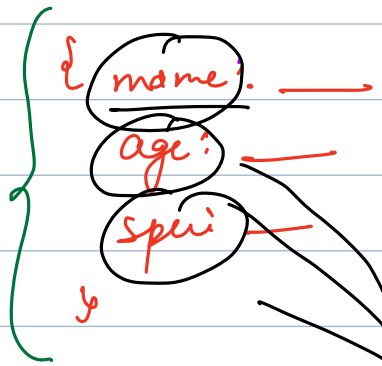
List<Object>

List<Animal>

List<User>

List



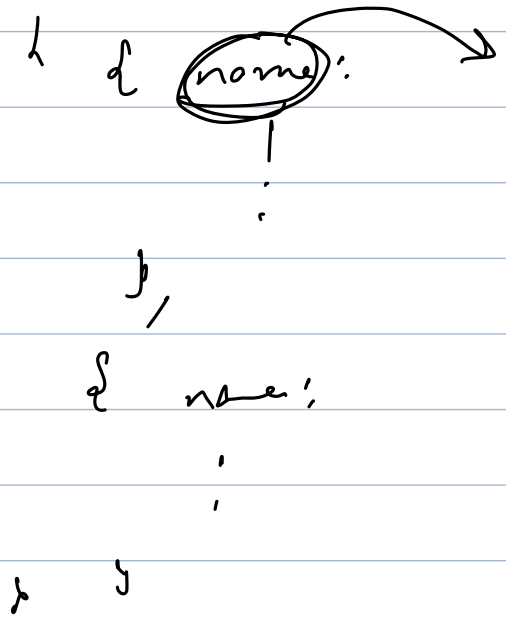


request.get(url, Animal.class)

Animal {
name
age
spui
}

request.get(url, List<Animal>.class)

List<Object>



List<Object>