**class** Vec<T> {

**private** Object[] elems = **new** Object[16];

**private** int end = 0;

**public** void add**(** ~~Object~~ T e**) {**

**if(**end == elems.length**) {** ... **}**

elems[end] = e;

end=end+1;

**}**

**public** ~~Object~~ T  **get(int index) {**

**if(index >= end) { throw ... }**

**else {**

**return (T) elems[index];**

**}**

**}**

**}**

Vec<Cat> v = **new** Vec<Cat>**()**;

v.add**(new** Cat**())**;

Cat c = ~~(Cat)~~v.get(0); done have to cast

How can we say v is a Vec of Cats?

**Java Generics**

* History

– Introduced in Java1.5

– Similar to C++ templates, but actually quite different as well!

• Before Java generics:

– **Can only say things like: ‘v’ is a Vector of Objects**

– Then, can put any Object into ‘v’ without restriction

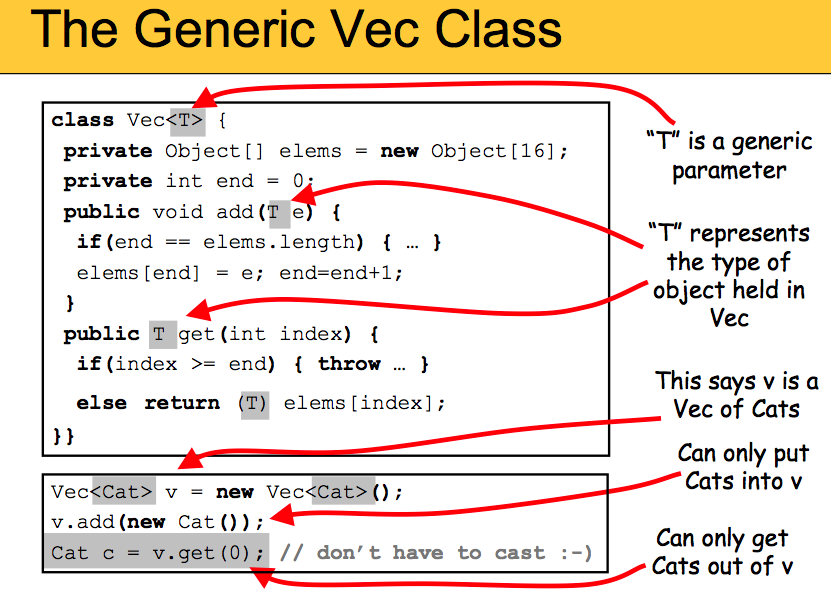
– With a Vector of just Cats, have to cast Objects to Cats

• With Java Generics:

– **Can say things like: ‘v’ is a Vector of Cats**

– Then, can only put Cats into ‘v’

– And, can only get Cats out of ‘v’–no casting required!



**class** Pair<FIRST,SECOND> {

**private** ~~Object~~ FIRST first;

**private** ~~Object~~ SECOND second;

**public** Pair(~~Object~~ FIRST f, ~~Object~~ SECOND s) {

first = f; second = s;

}

**public** ~~Object~~ first() { **return** first; }

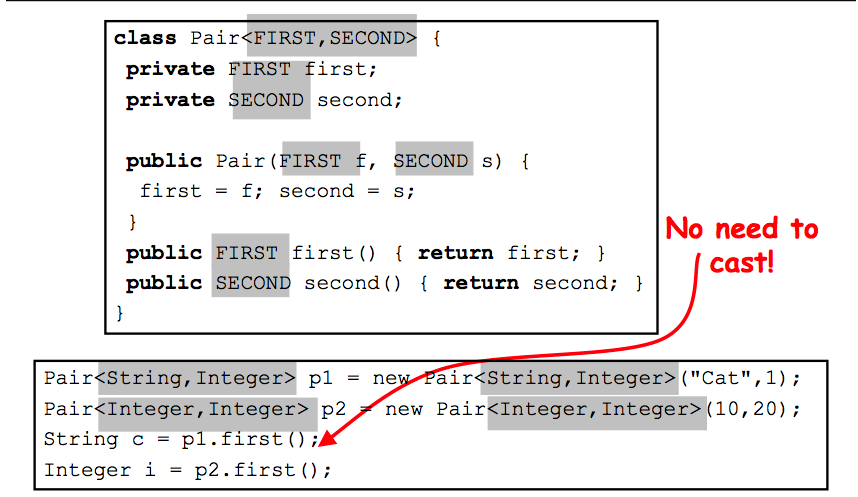
**public** ~~Object~~ second() { **return** second; }

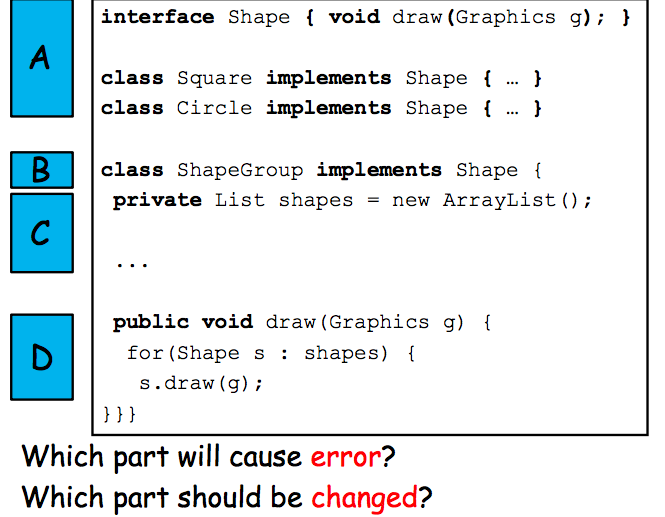
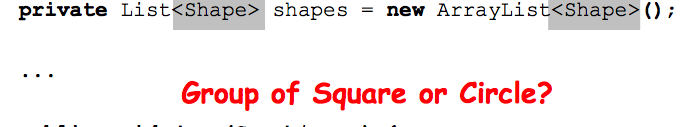
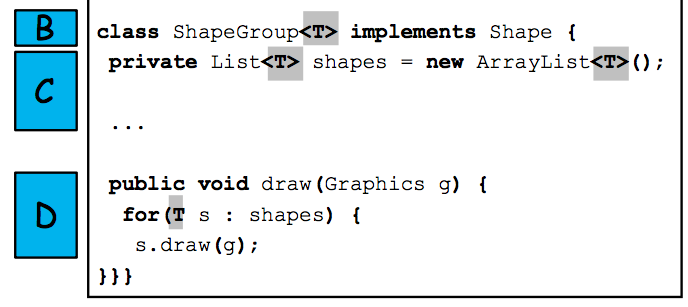
}

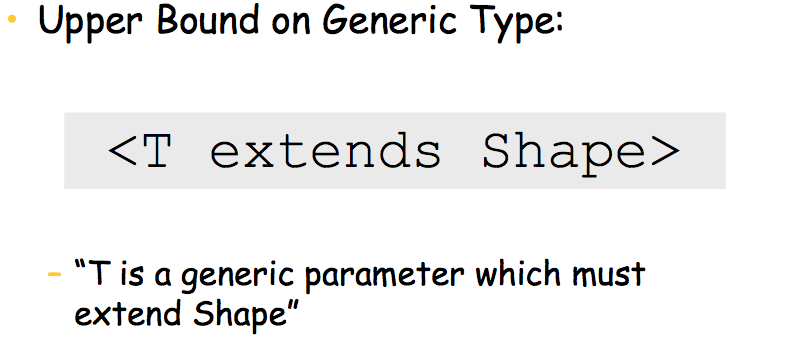
Pair <FIRST,SECOND> p1= **new** Pair<FIRST,SECOND> ("Cat",1);

Pair <FIRST,SECOND> p2= **new** Pair<FIRST,SECOND> (10,20);

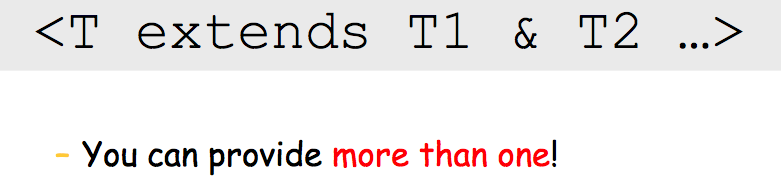
String c = ~~(String)~~ p1.first();

Integer i = ~~(Integer)~~ p2.first();





**Type is the name of class or interface**

****

<T2 extends List<T1>>

– You can express non-trivial ones!

