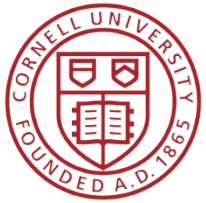
Photo credit: Andrew Kennedy

Generics and Collections

CS 2112

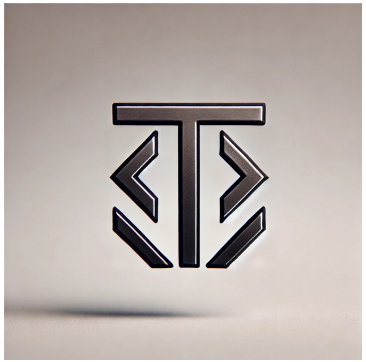


Generics

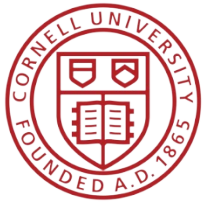
Goal: define class that **takes a type as a parameter.**

Ex. List<Integer>, Map<String>
Motivation: make Collections safer to use

```
public class MyListForAnything<T> {
        // The T can be any type
        public void add(T val) {
                ...
        }
}
```



<T> generated by GPT
Clearly no understanding!



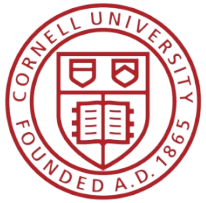
Generics – Type Erasure

Idea: Once your code is compiled, the **generic types are erased**.

Limitations:

1. Constructors of T cannot be used; we cannot write new T().
2. We cannot write new T[n].
3. We can't use instanceof to test what T is.

⇒ Can't do these things -- At run time, T is not known!



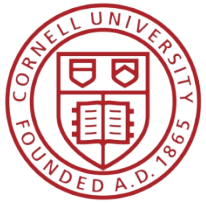
Pre compilation vs Post compilation

```java
public static void typeErasureExample() {
    List<String> a = new ArrayList<String>();
    a.add("foo");
    String foo = a.get(0);
}
```

```java
public static void typeErasureExample() {
    List<String> a = new ArrayList();
    a.add("foo");
    String foo = (String)a.get(0);
}
```

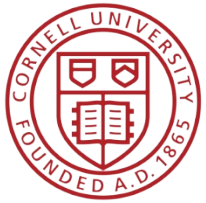
Creating ArrayList of Strings,
but the generic type String is erased

Now, to get something from the list,
no longer know what's inside, need to cast



Arrays of Generics

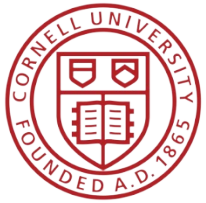
```
class Example {
        public static T[] makeArray(int length) {
                return new T[length];  //Illegal!
        }
}

class ExampleWorkaround {
        public static <T> T[] makeArray(int length) {
                return (T[]) (new Object [length]); //Legal, but not always safe
        }
}
```



Arrays of Generics

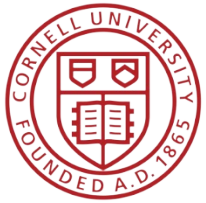
Workaround: Raw Types

HashSet<T>[] sets = **new** HashSet[n];



Collections

Idea: when a list of generic types is needed, use a collection instead of an array!

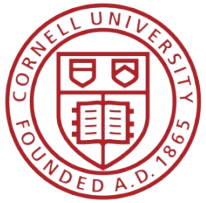


Collections

Key functionalities: put things into, take things out of, test for membership in, and iterate over.

```
interface Collection<T> {
    boolean contains(T x);
    boolean remove(T x);
    boolean add(T x);
}
```

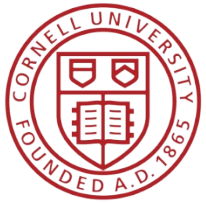
```
Collection c = ...;
c.add(new Integer(2)); // no check
for (Object o : c) {
    Integer i = (Integer) o;
    // use i for something
}
```



Collections

Interface that describes collections of items.

Ex: ArrayList, LinkedList, HashSet



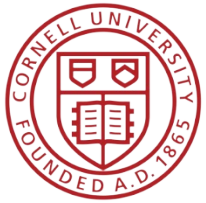
Collection Methods

boolean add(E e)
- adds an element of type E, returns whether the collection added the element

boolean contains(Object o)
- returns true if o is in the collection

boolean remove(Object o)
- removes the element equal to o if it is present, returns whether something was actually removed

int size()
- returns the number of elements in the collection

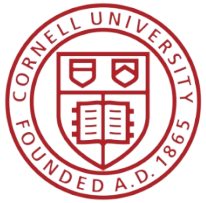


Iterating Collections

You can also iterate through elements of a collection with a foreach loop

Ex:

```
ArrayList<Integer> nums = ….;
        for (Integer i : nums) {
                //do something in the loop
}
```



Collections and Subtyping

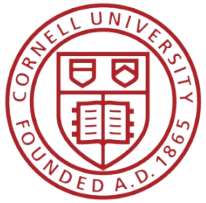
```
LList<T> implements Collection<T>
```

```
    LList<T>  <: Collection<T>
      String  <:  Object
LList<String> <: LList<Object>
```

✅

✅

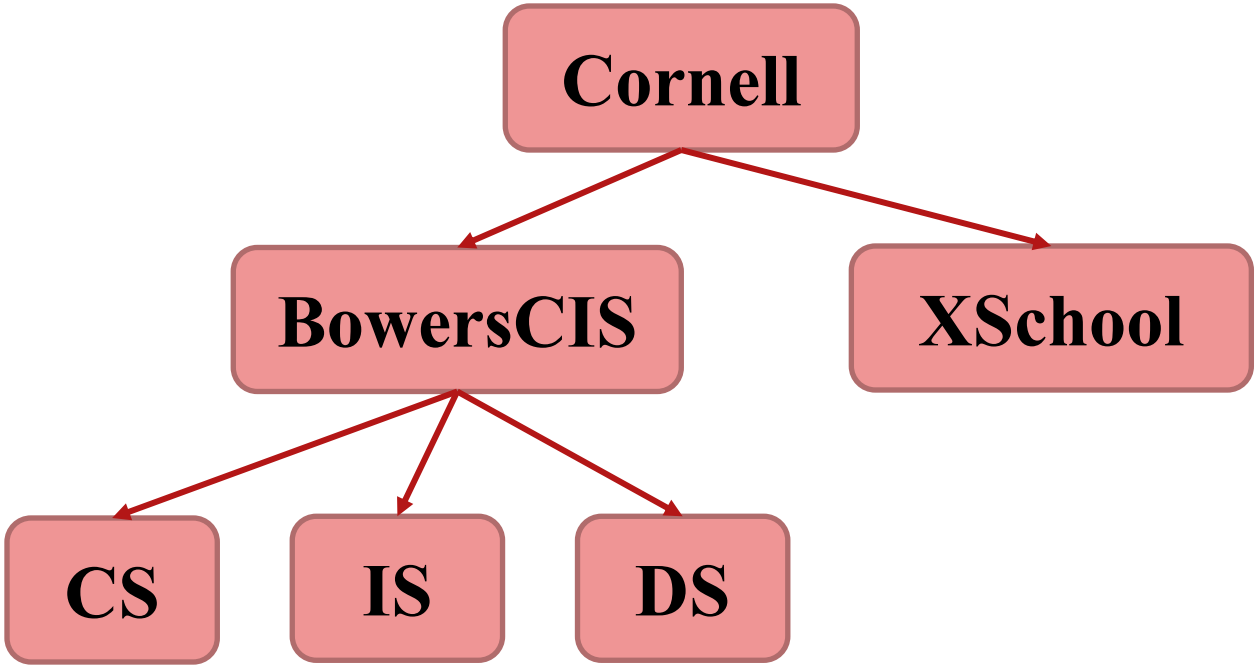
❌

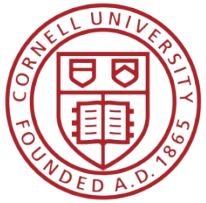


Wildcards

There are three wildcard operators we can use to represent groups of types in generics.

- **List <?>**
  - A List of any type
- **List <? extends A>**
  - A List of any subtype of A
  - Upper Bound
- **List <? super A>**
  - A List of any supertype of A
  - Lower Bound


Cornell
BowersCIS
XSchool
CS
IS
DS

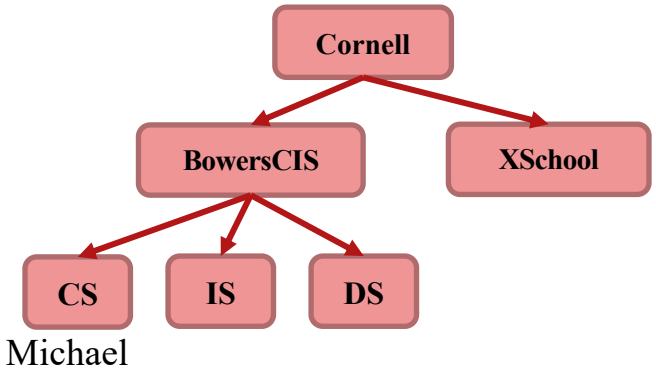


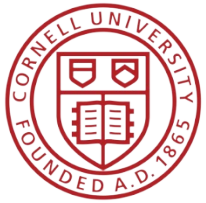
Example

```
List<?> unbound = new ArrayList<>();
List<? extends BowersCIS> ext = new ArrayList<>();
List<? super BowersCIS> sup = new ArrayList<>();
```

```
CS Michael = new CS();
unbound.add(Michael);
ext.add(Michael);
sup.add(Michael);
```

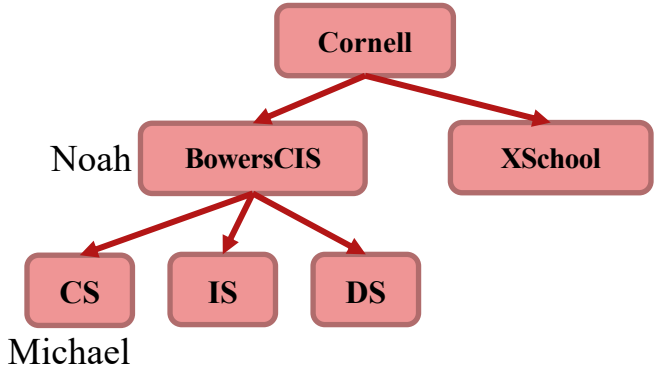
❌
❌
✅





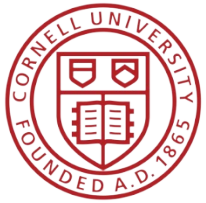
Example

```
List<CS> csStudents = new ArrayList<>();
csStudents.add(Michael);
List<?> unbound = csStudents;
BowersCIS Noah = new BowersCIS();
```



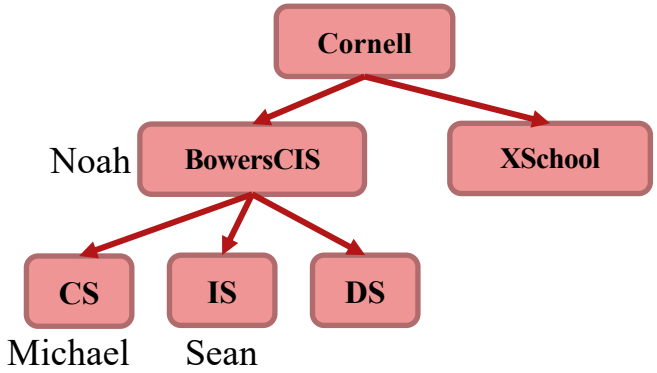

```
unbound.add(Noah);
```

This would be adding someone potentially not in CS to a CS student list

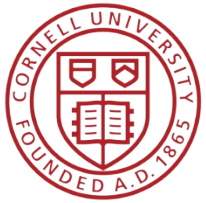


Example

```
List<IS> isList = new ArrayList<>();
IS Sean = new IS();
isList.add(Sean);
List<? extends BowersCIS> ext = isList;
CS Michael = new CS();

ext.add(Michael);
```

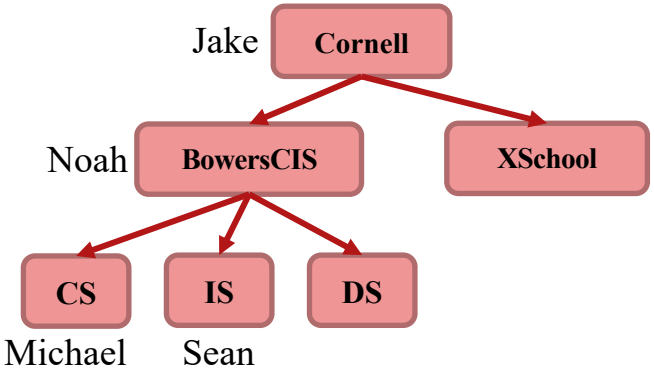


This would be adding someone in CS to an IS student list

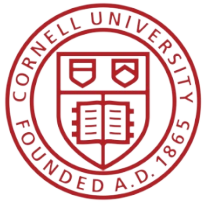


Example

```
List<Cornell> cornellStudents = new ArrayList<>();
Cornell Jake = new Cornell();
cornellStudents.add(Jake);
List<? super BowersCIS> sup = cornellStudents;
CS Michael = new CS();

sup.add(Michael);
```



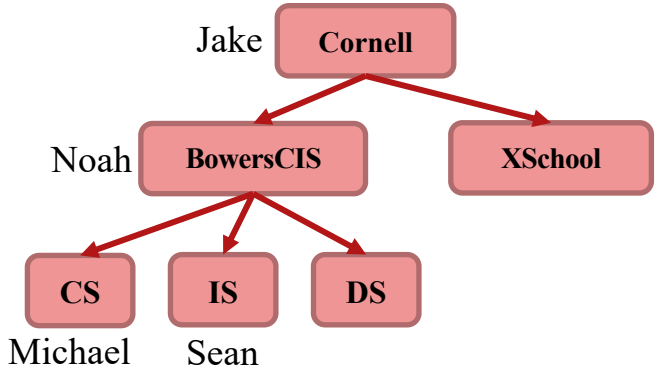
A CS student is always a Cornell Student

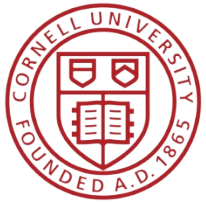


Group Exercise

```
List<BowersCIS> cisStudents= new ArrayList<>();
cisStudents.add(Noah);
List<?> unbound = cisStudents;
List<? extends BowersCIS > ext = cisStudents;
List<? super BowersCIS > sup = cisStudents;

BowersCIS i = unbound.get(0);
BowersCIS i = ext.get(0);
BowersCIS i = sup.get(0);
```

❌
✅
❌

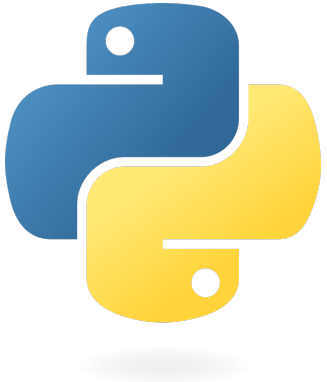




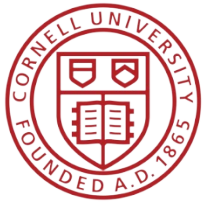
Raw Types

Idea: you don't actually have to specify a type!

```
List randomStuff = new ArrayList<>();
randomStuff.add(Michael)
randomStuff.add(Sean)
randomStuff.add(Noah)
randomStuff.add(Jake)
randomStuff.add(2112)
randomStuff.add(true)
randomStuff.add(null)
```



All works, but very dangerous!
Java will warn you.

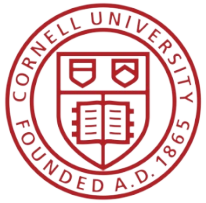


Array of Collections

This is probably the only case in which you would want to use a raw type!

```
ArrayList<Integer>[] arrListArr = new ArrayList<Integer>[10];
ArrayList<Integer>[] arrListArr2 = new ArrayList[10];
```

❌
✅



Group Exercise: is this legal?

```
Set<?> setOfUnknownType = new LinkedHashSet<String>();
setOfUnknownType = new LinkedHashSet<Integer>();
```

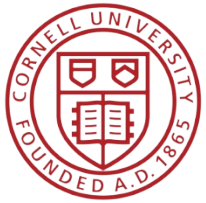
?

```
Set<String> setOfString = new HashSet<String>();
Set<Object> setOfObject = new HashSet<String>();
```

?

```
Set<? extends Number> set = new HashSet<Integer>();
set = new HashSet<Float>();
```

?



Exercise: is this legal?

```
Set<?> setOfUnknownType = new LinkedHashSet<String>();
setOfUnknownType = new LinkedHashSet<Integer>();
```

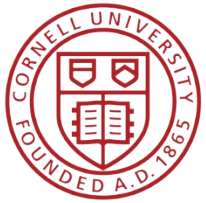
✅

```
Set<String> setOfString = new HashSet<String>();
Set<Object> setOfObject = new HashSet<String>();
```

❌

```
Set<? extends Number> set = new HashSet<Integer>();
set = new HashSet<Float>();
```

✅



Thank You && Good luck on A3!