JAVA GENERICS

FLEXIBILITY AND COMPLEXITY

Dr Lee A. Christie

 @javaxnerd

 leechristie.com

 lee@leechristie.com

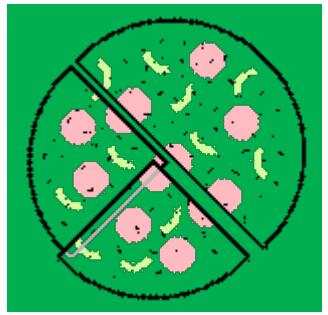
ON THE ORIGIN OF GENERICS

Based on C++ templates

It started with Pizza

Pizza Language

- Superset of Java
- Adds Generics → Java 5.0
- Adds Functional Features
- Abandoned 2002 → Scala, Java 8



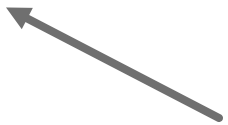
(http://pizzacompiler.sourceforge.net/)

ON THE ORIGIN OF GENERICS

```
List lst = new LinkedList();
lst.add("Hello");

String str = (String) lst.get(0);
```

Consumes Objects



Produces Objects,
typecast needed

ON THE ORIGIN OF GENERICS

```
List<String> lst = new LinkedList<String>();
lst.add("Hello");

String str = lst.get(0);
```

Consumes Strings
(Homogenous)

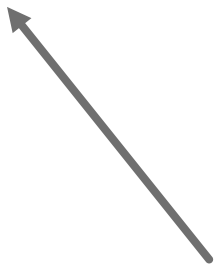
Produces Strings,
typecast not needed

compile-time
type safety

ON THE ORIGIN OF GENERICS

```
List<String> lst = new LinkedList<>();
lst.add("Hello");

String str = lst.get(0)
```



Type inference
(Java 7's Project Coin)

NOT JUST COLLECTIONS

From concurrency:

`Callable`	a task that returns a result
`Future`	the result of a future computation
`ThreadLocal`	… provides thread local variables
`AtomicReference`	an object reference that may be updated atomically

From functional operations:

`Optional`	a container object which may or may not contain a non-null value
`Function`	a function that accepts one argument and produces a result
`Stream`	A sequence … supporting sequential and parallel … operations
`Consumer`	… accepts a single input argument and returns no result

SOME TERMINOLOGY

Generic type:

`List<E>`

Read “list of E”

E is the formal parameter

Parameterized type:

`List<Integer>`

Read “list of integer”

Raw type:

`List`

Tells compiler: “I don’t want compile–time type safety”

NEVER USE THIS

(break for coding demo 1)

QUIZ TIME

```
1    Object[] arr = new String[2];
2    arr[0] = "Hello";
3    arr[1] = Integer.valueOf(42);
4    System.out.println(Arrays.toString(arr));
```

ArrayStoreException

A. prints "[Hello, 42]"

B. compile error (line 1)

C. compile error (line 3)

D. run time exception

QUIZ TIME

```
1    LinkedList<Object> lst = new LinkedList<String>();
2    lst.add("Hello");
3    lst.add(Integer.valueOf(42));
4    System.out.println(lst);
```

incompatible types: `LinkedList<String>` cannot be converted to `LinkedList<Object>`

A. prints "`[Hello, 42]`"

B. compile error (line 1)

C. compile error (line 3)

D. run time exception

LISKOV SUBSTITUTION PRINCIPLE (LSP)

"Let φ(x) be a property provable about objects x of type T. Then φ(y) should be true for objects y of type S where S is a subtype of T."

(Liskov and Wing, 1994)

Informally:

If S is a T, we should be able to treat an S as a T.

Violates LSP (We cannot treat a `String[]` as an `Object[]`)

```
Object[] arr = new String[2];
```

Respects LSP (by not compiling):

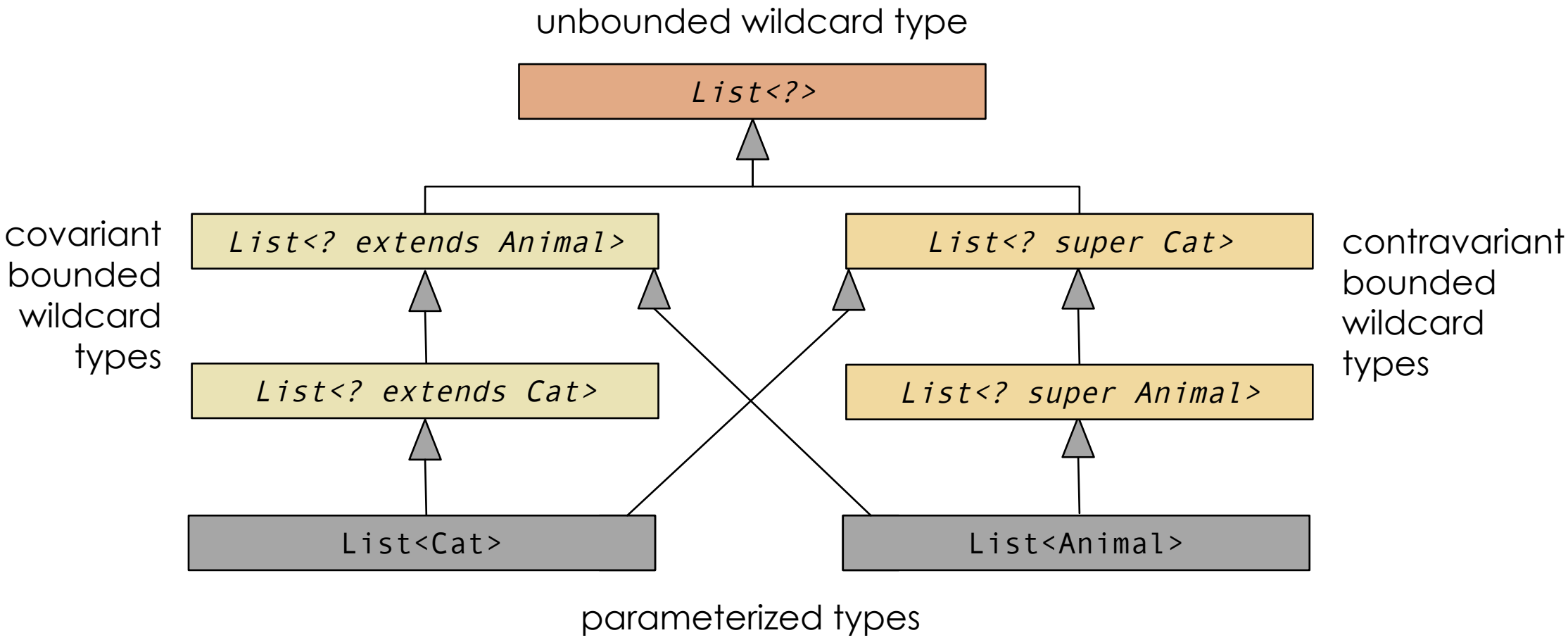
```
LinkedList<Object> lst = new LinkedList<String>();
```

LISKOV SUBSTITUTION PRINCIPLE (LSP)

SOLID Motivational Posters
Derick Bailey, CC BY-SA 3.0

(break for coding demo 2)

WILDCARDS


unbounded wildcard type
List<?>
covariant bounded wildcard types
List<? extends Animal>
List<? super Cat>
contravariant bounded wildcard types
List<? extends Cat>
List<? super Animal>
List<Cat>
List<Animal>
parameterized types

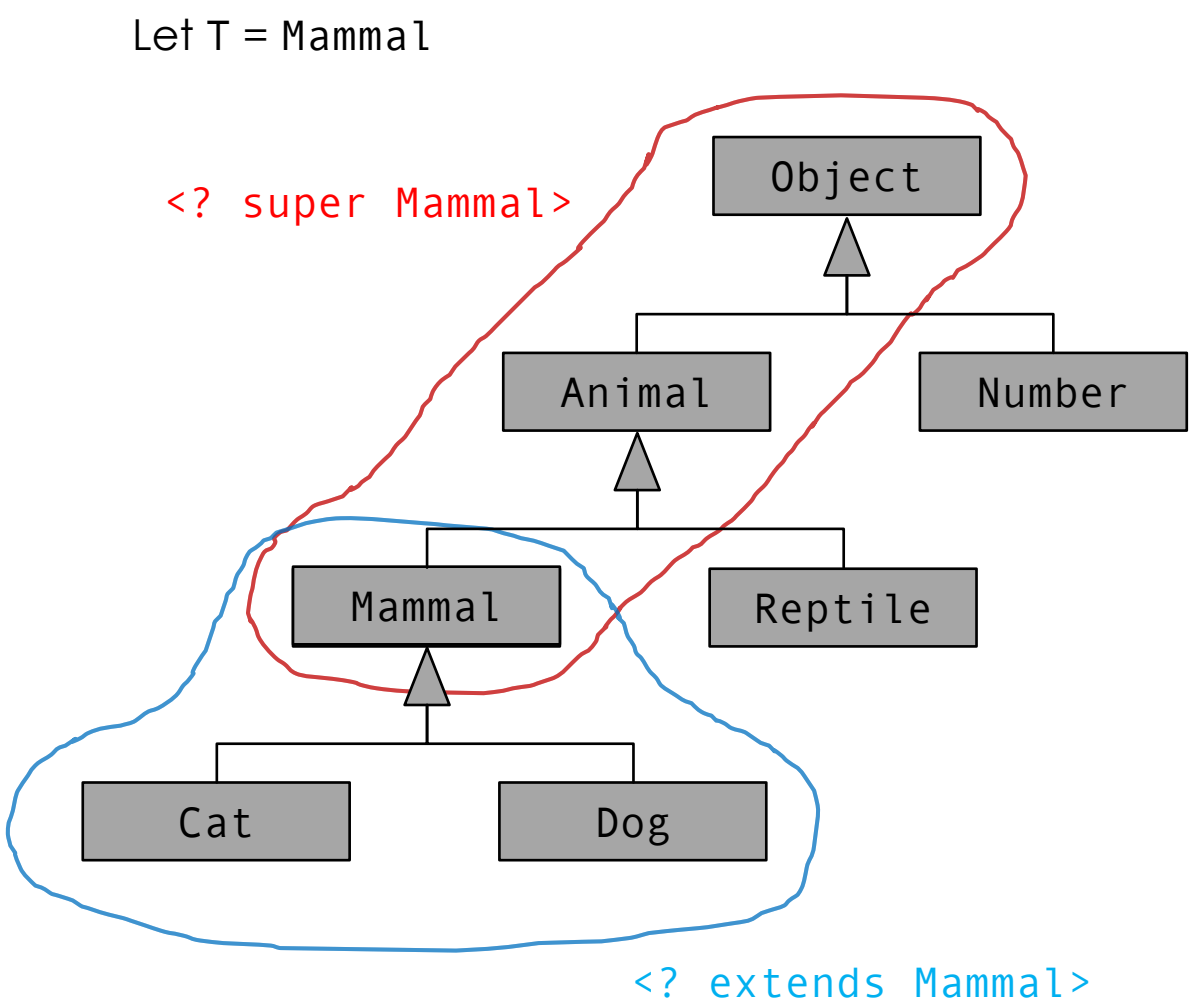
WILDCARDS

	Produces	Consumes
List<?>	Object	*(forbidden)*
List<? extends Animal>	Animal	*(forbidden)*
List<? extends Cat>	Cat	*(forbidden)*
List<? super Cat>	Object	Cat
List<? super Animal>	Object	Animal
List<Cat>	Cat	Cat
List<Animal>	Animal	Animal
List<Object>	Object	Object

PECS (PRODUCER EXTENDS, CONSUMER SUPER)

		Produces T Instances: Yes	Produces T Instances: No
Consumes T Instances	Yes	`<T>` (invariant)	`<? super T>` (contravariant)
Consumes T Instances	No	`<? extends T>` (covariant)	`<?>` (independent)

(Bloch 2009)



(break for coding demo 3)

ERASURE

Compile Time

```
void check(T)
void check(T[])
void check(Collection<? extends T>)
void check(Collection<String>)
```

Run Time

```
void check(Animal)
void check(Animal[])
void check(Collection)
void check(Collection)
```

Generics do not exist (at run time)

ERASURE

"… Pre-existing code must work on the new system. This implies ... upward compatibility of the class file format …"

"… Upward source compatibility. It should be possible to compile essentially all existing Java language programs with the new system."

"**The proposed extension has absolutely no effect on the Java Virtual Machine specification**"

(JSR-14 Public Draft)

ERASURE

"Supporting generic types at run time seems undesirable for the following reasons:

Lack of experience with such constructs in widely used languages

Burden of extensive VM changes on vendors throughout the industry

Increased footprint on small devices

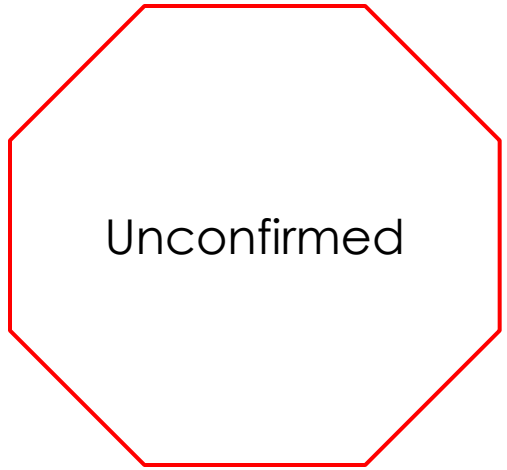
Decreased performance for generic methods

Compatibility"

Questionable?
(C# generics are faster than non-generics)

(JSR-14 Public Draft)

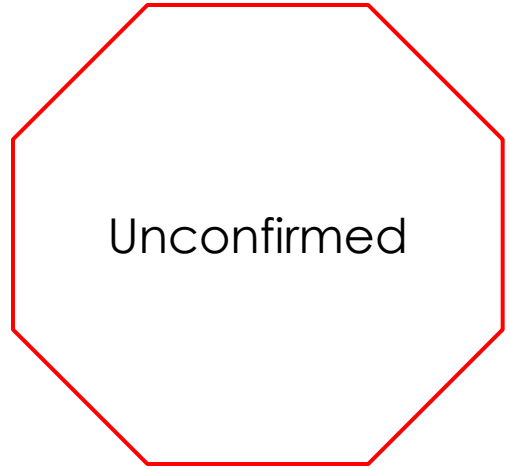
PROJECT VALHALLA



Targeting Java 10:

- Reified generics
- Generic specialization
- Value types
- Improved `volatile` support

ERASED/REIFIED GENERICS



Currently Not Possible (Java 5.0 to present, Erased)

(might be allowed in Java 10, Refied):

```
new List<Integer>[];
```

```
new T[];
```

```
if (foo instanceof List<Integer>) {…}
```

```
if (foo instanceof T) {…}
```

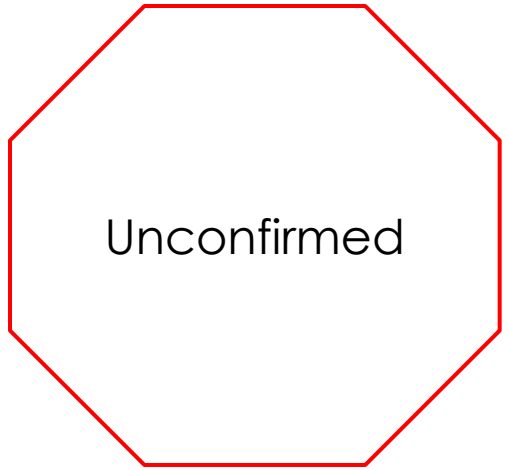
```
Class<?> = List<Integer>.class;
```

```
Class<?> = T.class;
```

```
void foo(List<String> x) {…}
void foo(List<Integer> x) {…}
```

```
void bar(T x) {…}
void bar(S x) {…}
```

GENERIC SPECIALIZATION



Currently (Java 5.0 to present):

`List<Integer>` extends `List<?>`

`?` always refers to `Object` subtypes

`List<int>` does not exist!!!

One day? (Java 10?):

`List<int>` extends `List<any ?>`

`More efficient collections`

`Cleaner functional code`