What are Generics?

e.g. Generics, Generic Programming,
Generic Types, Generic Methods

Defining the idea Behind Java Generics

Data Types are now used as Type Parameters

Defining the idea Behind Java Generics

Generic Types:

"generic class or interface which is parameterized over types"

Generic Methods:

"Method featuring type parameters"

https://docs.oracle.com/javase/tutorial/java/generics/types.html
http://www.angelikalanger.com/GenericsFAQ/FAQSections/ParameterizedTypes.html#What is a parameterized (or generic) type?
http://www.angelikalanger.com/GenericsFAQ/FAQSections/ParameterizedMethods.html#What is a parameterized (or generic) method?

Why Bother?

Simplify notations (?) by eliminating need for explicit casting every time

```
List list
                = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

Without Generics

With Generics

```
List<String> list
        = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

Why Bother?

Stronger Type Checking

Less bugs

```
List list = new ArrayList();

list.add("hello");
list.add(new Integer(42));

String s = (String) list.get(0);
Integer s = (Integer) list.get(1);
```

What happens if

Runtime Exception

```
Integer s = (String) list.get(1);
```

Why Bother?

Encourages programmers to write generic classes

e.g.

- Originally meant to support Java Collections
  - A List of <Integer> has the same underlying logic as a List of <Double>
- Usable now by anyone to produce better code
  - Define the concept of List of <something>

What are Generic Types?

e.g. Generic Classes, Generic Interfaces

Is it possible to do Generic Programming w/o Generics?

```
public class Box {

    private Object object;

    public void set(Object object)
      { this.object = object; }

    public Object get()
      { return object; }
}
```

Consider this…

Let's try it out!

```
public static void main(String[] argv){

    Box b       = new Box();
    Integer n1  = new Integer(42);

    b.set(n1);

    Double n2   = b.get();
}
```

```
> "javac" Box.java
Box.java:11: error: incompatible types: Object cannot be converted to Double
    Double n2 = b.get();
                ^
1 error

> Process Exit Code: 1
> Time Taken: 00:01
```

How do we fix this?

Now, Let's fix it!

What is wrong with this?

Why #2 Bother?

```
public static void main(String[] argv){

    Box b        = new Box();

    Integer n1   = new Integer(42);

    b.set(n1);

    Double n2    = (Double)b.get();
}
```

```
> "javac" Box.java

> Process Exit Code: 0
> Time Taken: 00:01
```

Problem = Java does not know that this instance of Box is meant to store only Integer objects

New Plan!

We need a way to enable Java to check our code for such inconsistencies

What does this mean?

→ Rely on the compiler / runtime to

validate types usage

make necessary type castings

…

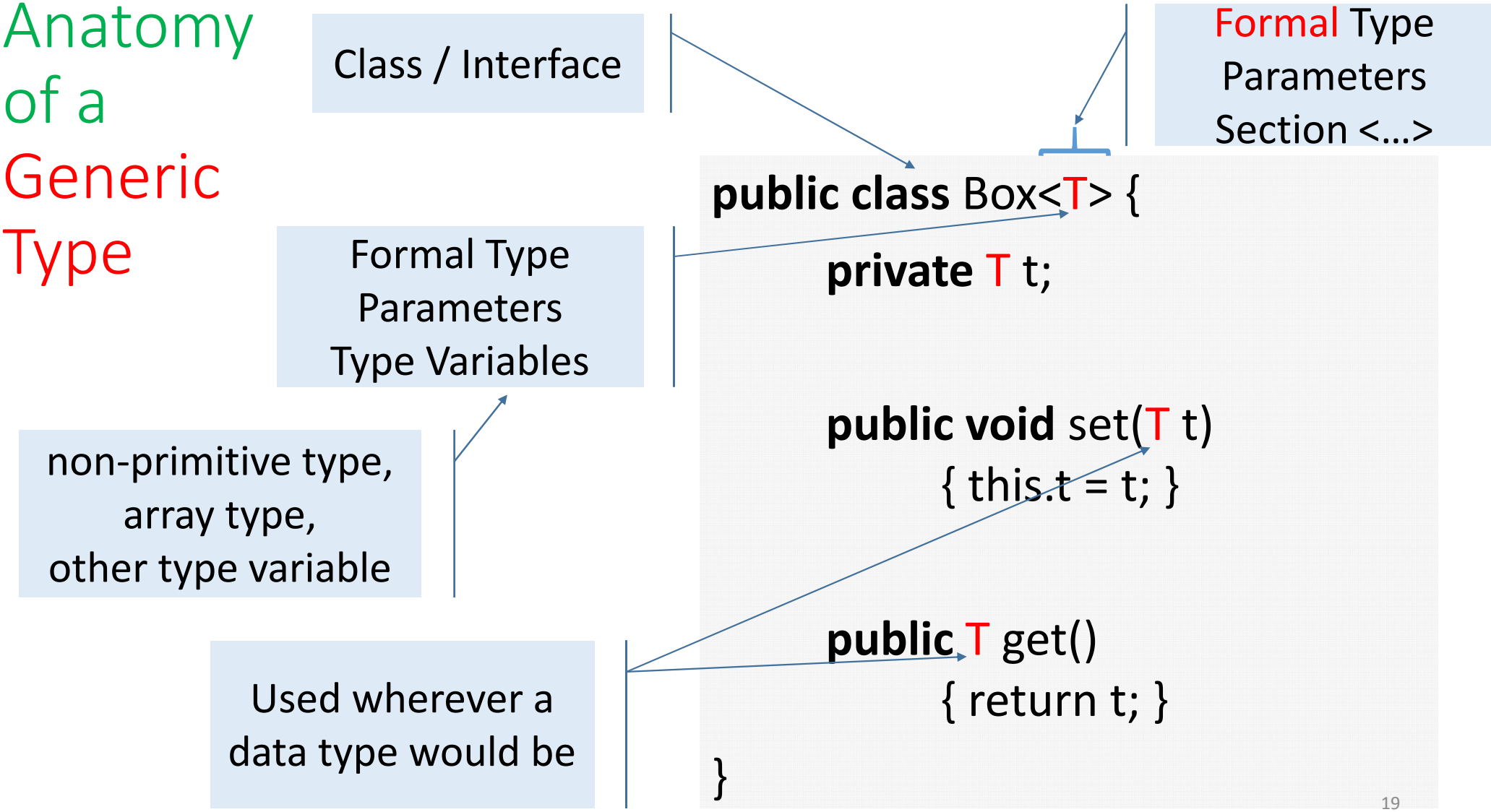
→ Add information to the code to help them do so

Introducing… Generic Types! See the difference?

```
public class Box {
    private Object object;

    public void set(Object object)
            { this.object = object; }

    public Object get()
            { return object; }
}
```

```
public class Box<T> {
    private T t;

    public void set(T t)
            { this.t = t; }

    public T get()
            { return t; }
}
```

Anatomy of a Generic Type



Naming Conventions for Formal Type Parameters

E - Element (used extensively by the Java Collections Framework)

K - Key

N - Number

T - Type

V - Value

S,U,V etc. - 2nd, 3rd, 4th types

How to Instantiate a Generic Class

Box<Integer> integerBox = new Box<Integer>();

Parameterized Type; i.e. instantiation of a generic type

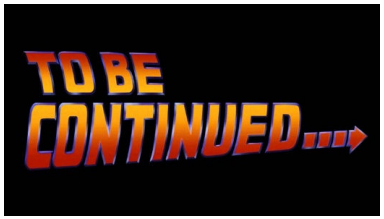
Box<Integer> integerBox = new Box<>();

Actual Type Argument

Since Java 7+
"Type Inference"

The Diamond

? What is Type Inference?



What happens if we do not use <> at all?

```
public class Box<T> {
    public void set(T t) { /* ... */ }
    // ...
}

Box rawBox = new Box();
```

```
public class Box<T> {
    public void set(T t) { /* ... */ }
    // ...
}

Box rawBox = new Box<Integer>();
```

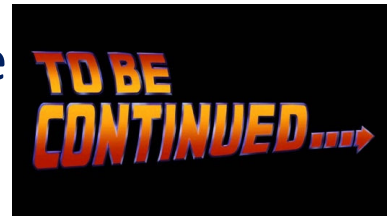
Raw types!

Legacy code < JDK 5.0

When using raw types, you essentially get pre-generics behavior — a Box gives you Objects

Type Erasure converts parameterized types into raw types

What is Type Erasure?



This little thing called…
…Backward Compatibility

Assigning a parameterized type to its raw type is allowed:

```
Box<String>     stringBox = new Box<>();
Box             rawBox    = stringBox;
```

Warning #1

assign parameterized type to its raw type → shiny

assign raw type to its parameterized type → warning

```
Box rawBox = new Box();
// rawBox is a raw type of Box<T>

Box<Integer> intBox = rawBox;
// warning: unchecked conversion
```

Warning #2

Do not use raw type object to invoke generic methods defined in the corresponding parameterized type:

```
Box<String> stringBox      = new Box<>();
Box rawBox                 = stringBox;

rawBox.set(8);
// warning: unchecked invocation to set(T)
```

What are these warnings trying to tell us?

```
Box<Integer> intBox = rawBox;
// warning: unchecked conversion

rawBox.set(8);
// warning: unchecked invocation to set(T)
```

Raw types bypass generic type checks
Back in the pre-generics days
The catch of unsafe code is deferred to runtime
Do not use raw types as a programmer!

Let's take an example…

```
public class WarningDemo {
    public static void main(String[] args){
        Box<Integer> bi;
        bi = createBox();
    }

    static Box createBox(){
        return new Box();
    }
}
```

PT – Parameterized Type

Assigning RT to PT

RT – Raw Type

```
Note: WarningDemo.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Get more details with -Xlint:unchecked

```
WarningDemo.java:4: warning: [unchecked] unchecked
conversion
found   : Box
required: Box<java.lang.Integer>
        bi = createBox();
                         ^
1 warning
```

We may suppress the warning with;

```
-Xlint:-unchecked
@SuppressWarnings("unchecked")
```