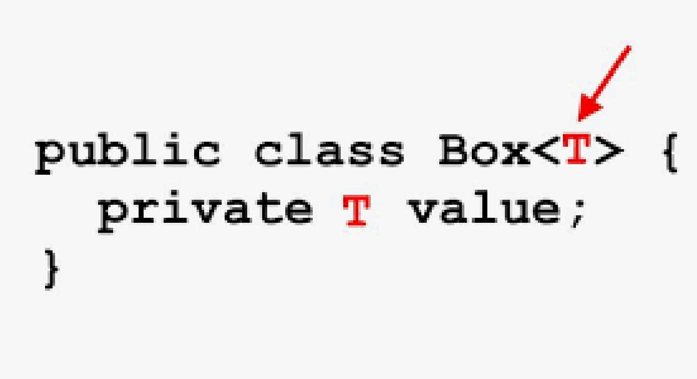
Generics

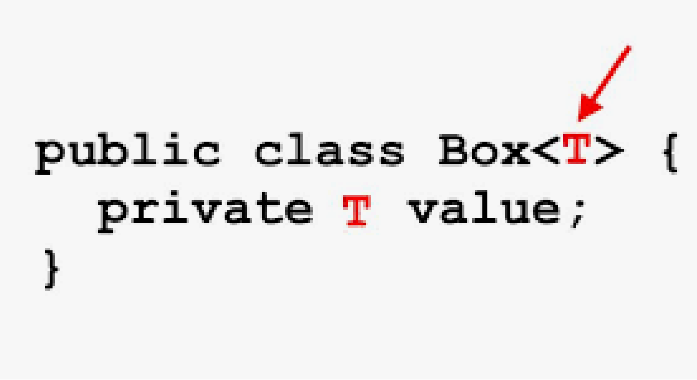


```
public class Box<T> {
  private T value;
}
```

Contents

- Introduction
- Basic Syntax
- Why Using Generics?
- Generic Classes and Interfaces
- Generic Methods
- Bounded Types
- Wildcards Generic Types

Generics



```
public class Box<T> {
  private T value;
}
```

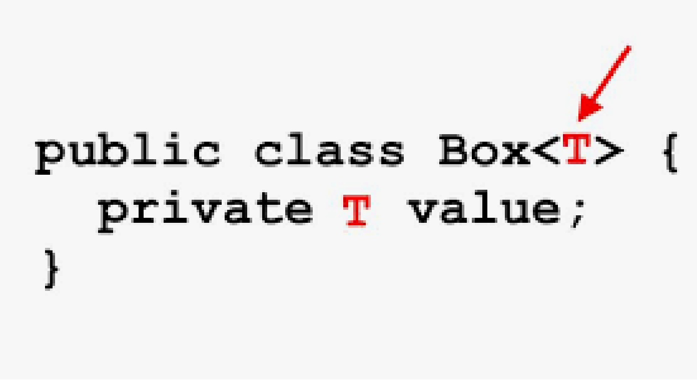
Introduction

- Generics introduced by Java JDK 1.5.
- Generics means parameterized types.
- Generics add the type safety.
- Generics does not work with primitive types (int, float, char, …).

Generics

Basic Syntax

T is a type parameter that will be replaced with a concrete type when the class is instantiated.

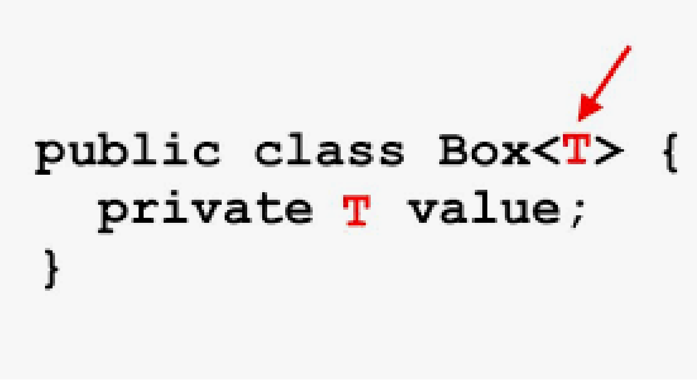


```
public class Box<T> {
  private T value;
}
```

```
public class Box<T> {
        private T t;
        public void set(T t) {
                this.t = t;
        }
        public T get() {
                return t;
        }
}
```

Generics

Why Generics?



```
public class Box<T> {
  private T value;
}
```

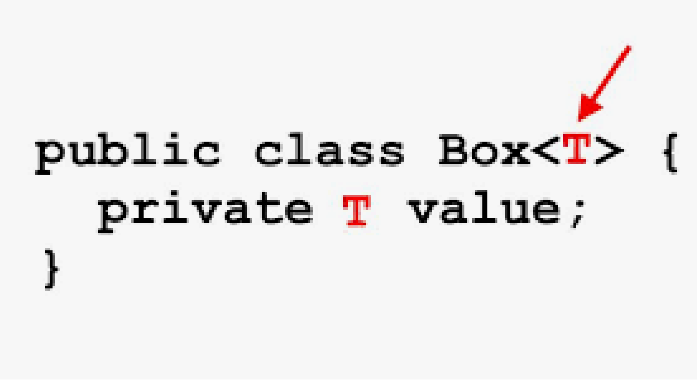
Generics can be achieved by specifying Object Type and using proper casting when it required

So why we use generics ??

By using Object , java Compiler doesn’t have info about the type of the data so ,

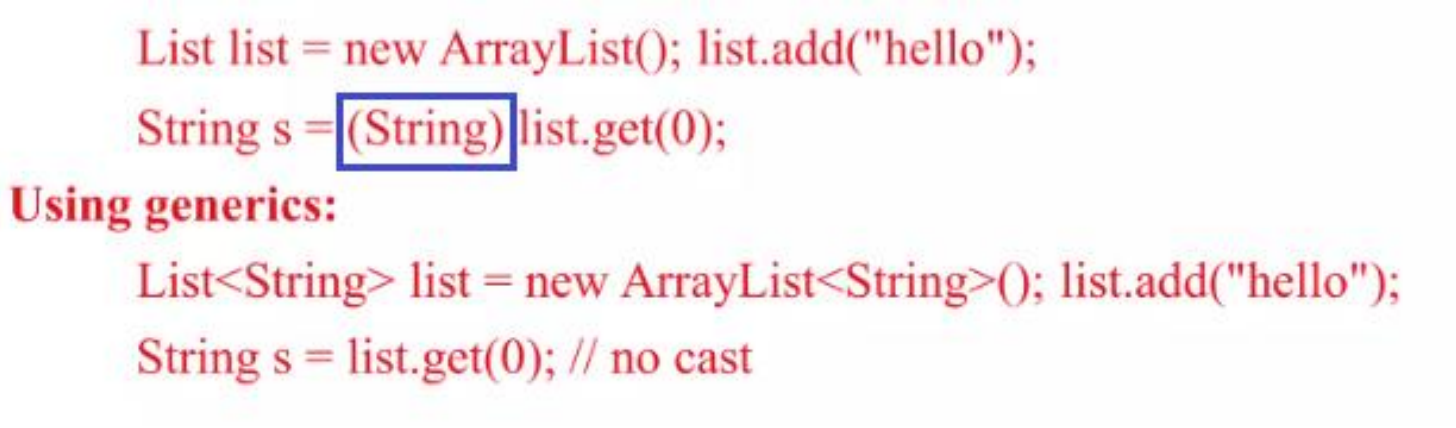
- Explicit Casts must be employed to retrieve the stored data.
- Several type mismatch errors can’t be found till runtime

Generics



```
public class Box<T> {
  private T value;
}
```

Why Generics?

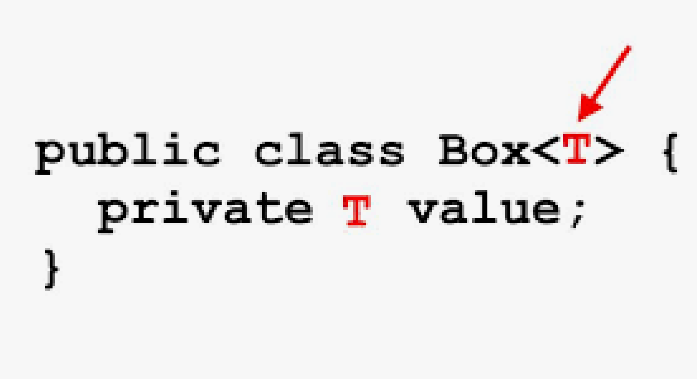


```
    List list = new ArrayList(); list.add("hello");
    String s = (String) list.get(0);
Using generics:
    List<String> list = new ArrayList<String>(); list.add("hello");
    String s = list.get(0); // no cast
```

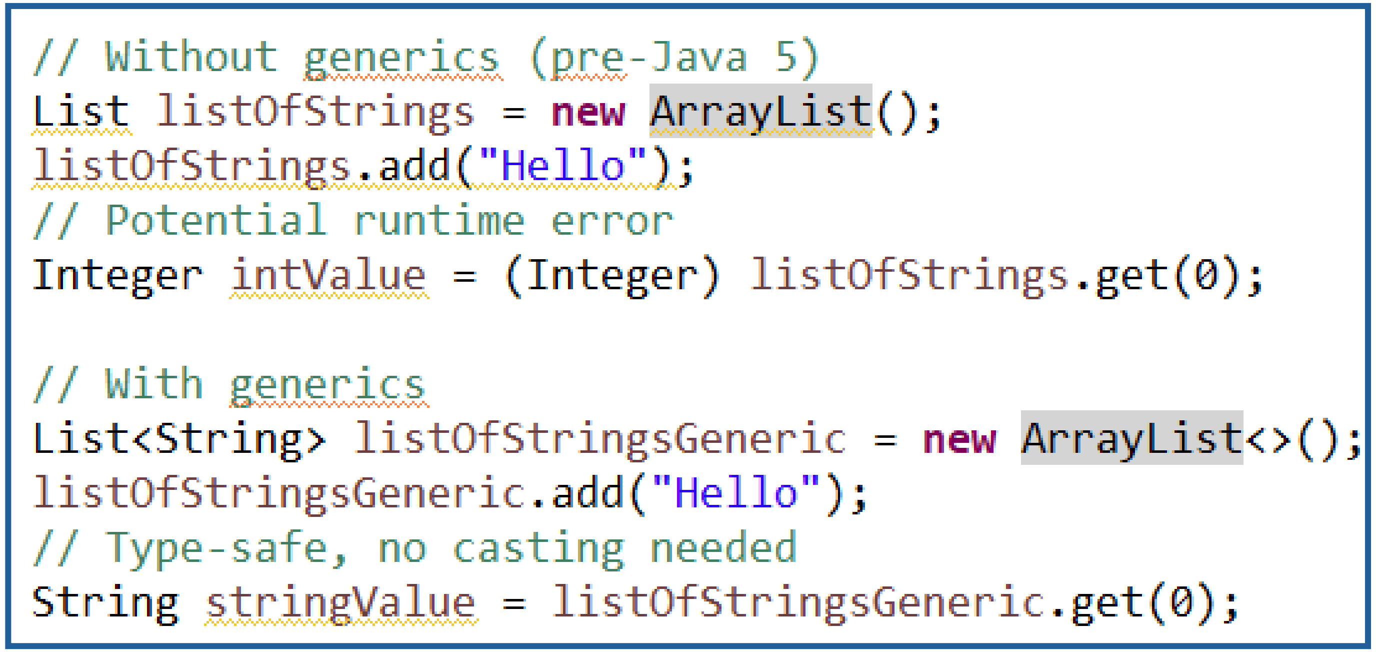
Generics

Why Use Generics?

Type safety:



```
public class Box<T> {
  private T value;
}
```

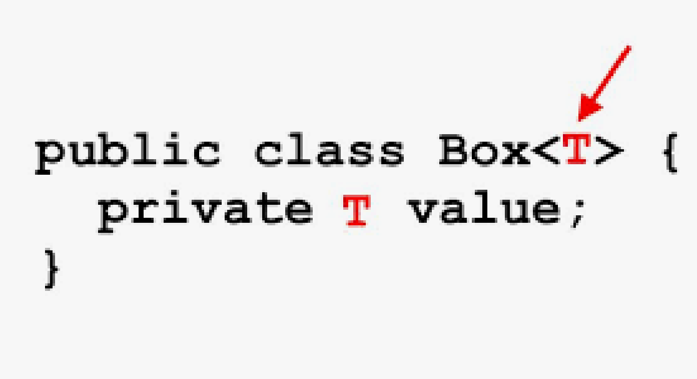


```
// Without generics (pre-Java 5)
List listOfStrings = new ArrayList();
listOfStrings.add("Hello");
// Potential runtime error
Integer intValue = (Integer) listOfStrings.get(0);

// With generics
List<String> listOfStringsGeneric = new ArrayList<>();
listOfStringsGeneric.add("Hello");
// Type-safe, no casting needed
String stringValue = listOfStringsGeneric.get(0);
```

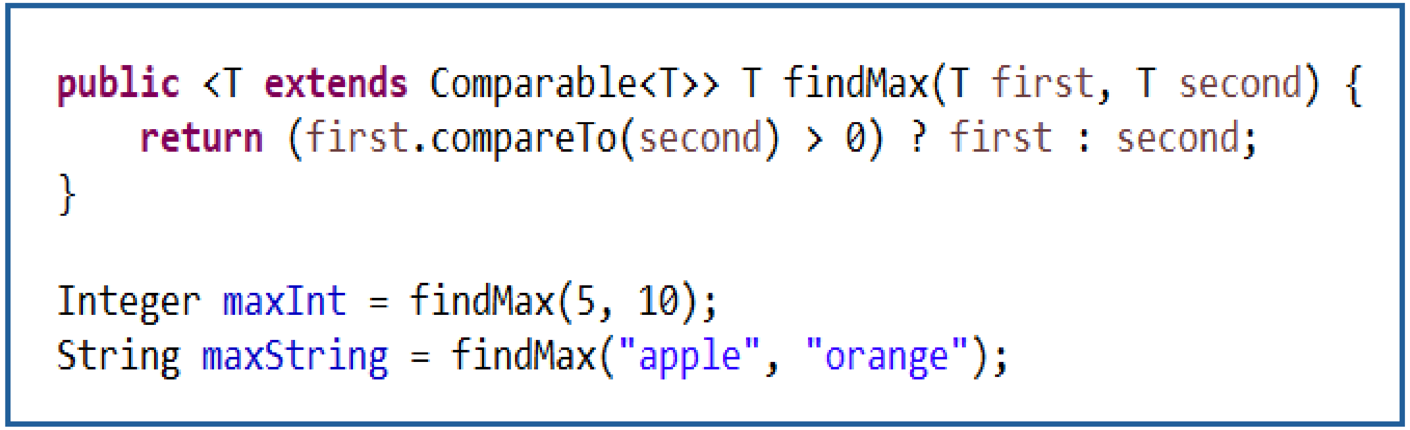
Generics

Why Use Generics?

code reusability

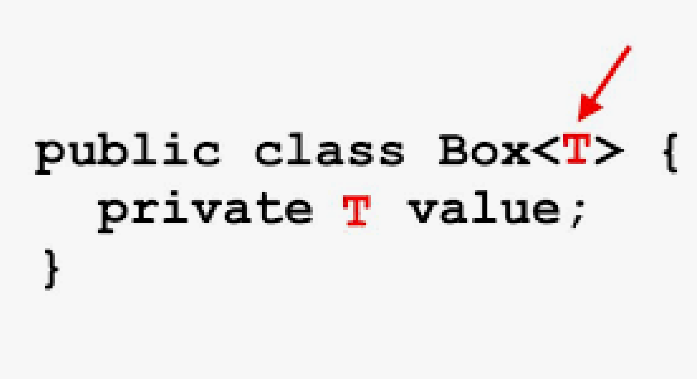


```
public class Box<T> {
  private T value;
}
```



```
public <T extends Comparable<T>> T findMax(T first, T second) {
    return (first.compareTo(second) > 0) ? first : second;
}

Integer maxInt = findMax(5, 10);
String maxString = findMax("apple", "orange");
```

Generics



```
public class Box<T> {
  private T value;
}
```

Generic is Working with references Data Types

Reference Types:

```
List<String> stringList = new ArrayList<>();
Map<Integer, String> map = new HashMap<>();
```

✓

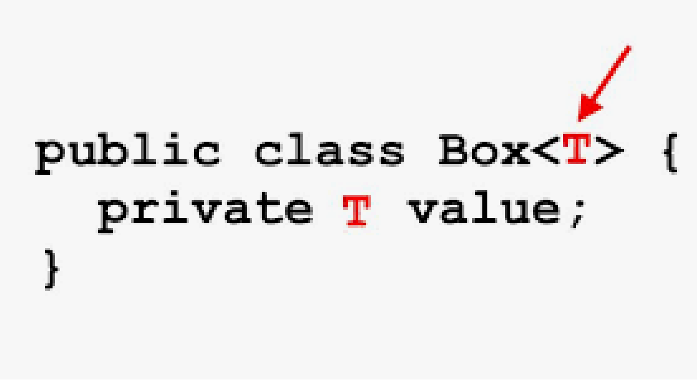
Primitive Types:

```
List<int> intList = new ArrayList<>(); // This is not allowed
```

✗

Generics

Generic is Working with references Data Types



```
public class Box<T> {
  private T value;
}
```

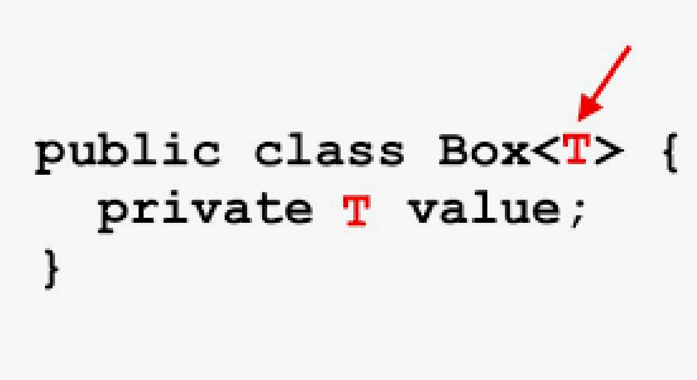
Wrapper Classes:

To use generics with primitive types, you must use their corresponding wrapper classes:

byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Generics

Generic is Working with references Data Types

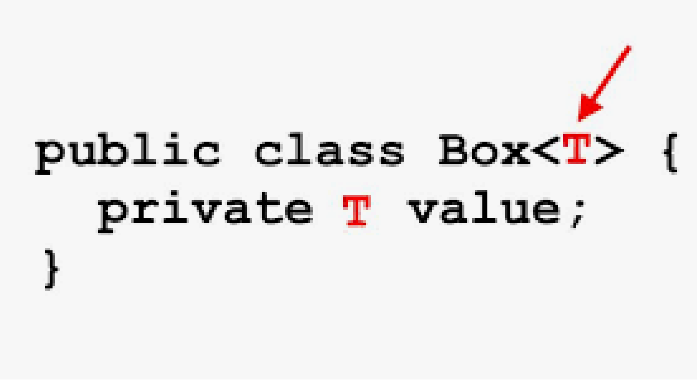


```
public class Box<T> {
   private T value;
}
```

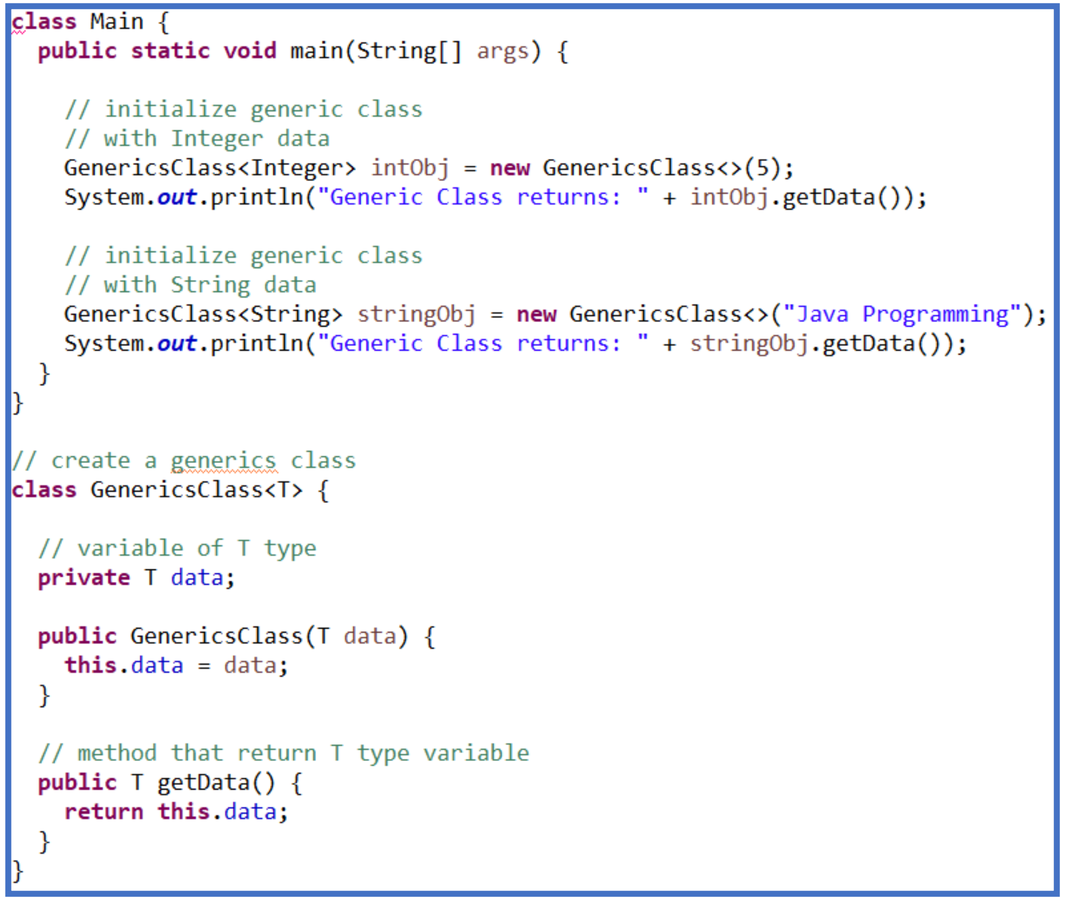
```
public class GenericExample {
   public static void main(String[] args) {
      List<Integer> numbers = new
ArrayList<>();
      numbers.add(10);
      numbers.add(20);
      numbers.add(30);

      for (Integer number : numbers) {
         System.out.println(number);
      }
   }
}
```

Generics

Java Generic Class



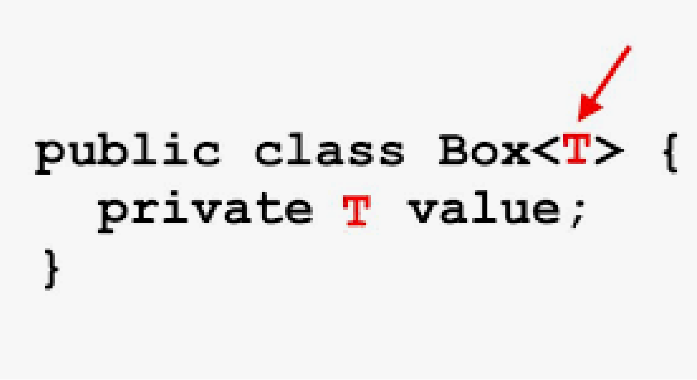
```
public class Box<T> {
  private T value;
}
```



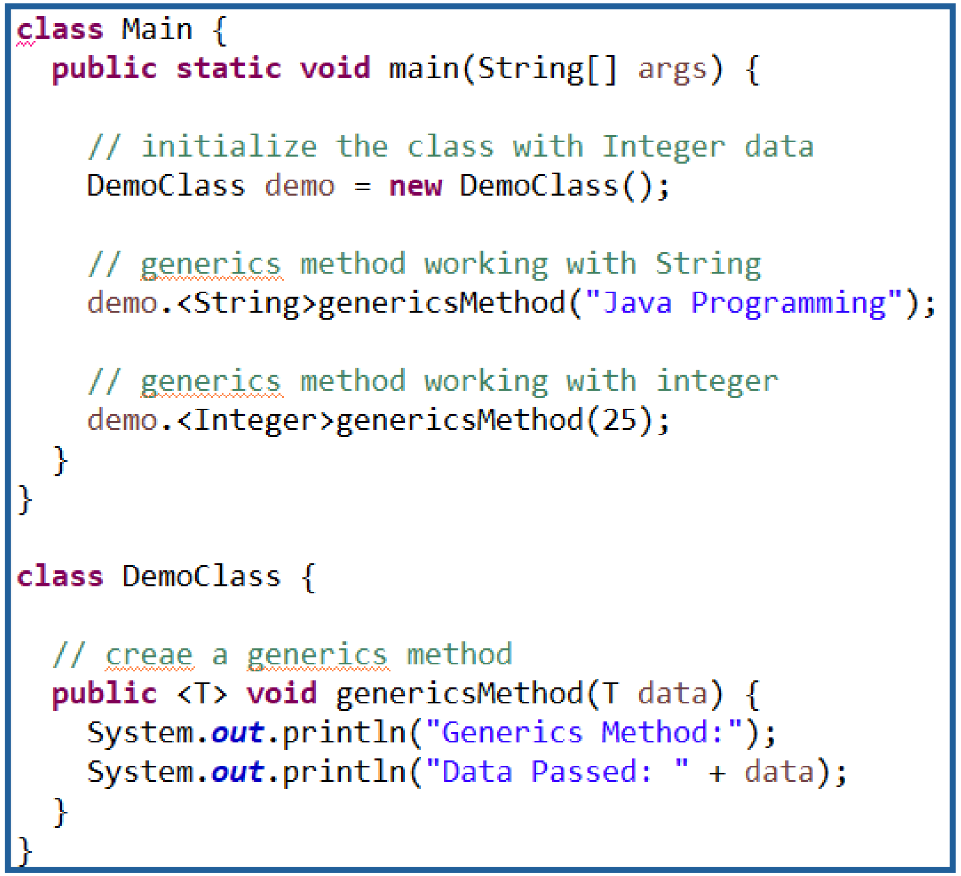
```
class Main {
  public static void main(String[] args) {

    // initialize generic class
    // with Integer data
    GenericsClass<Integer> intObj = new GenericsClass<>(5);
    System.out.println("Generic Class returns: " + intObj.getData());

    // initialize generic class
    // with String data
    GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");
    System.out.println("Generic Class returns: " + stringObj.getData());
  }
}

// create a generics class
class GenericsClass<T> {

  // variable of T type
  private T data;

  public GenericsClass(T data) {
    this.data = data;
  }

  // method that return T type variable
  public T getData() {
    return this.data;
  }
}
```

Generics

Java Generic Method



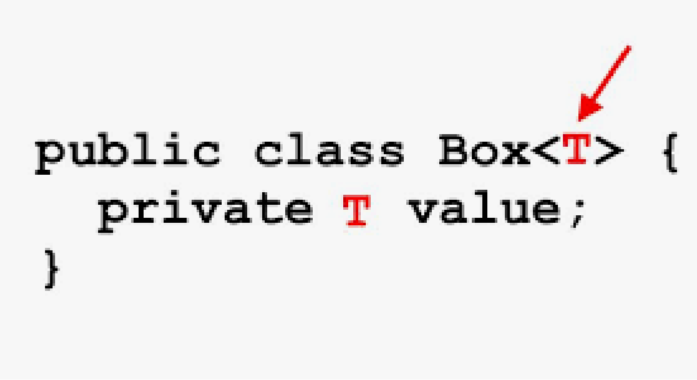
```
public class Box<T> {
  private T value;
}
```



```
class Main {
  public static void main(String[] args) {

    // initialize the class with Integer data
    DemoClass demo = new DemoClass();

    // generics method working with String
    demo.<String>genericsMethod("Java Programming");

    // generics method working with integer
    demo.<Integer>genericsMethod(25);
  }
}

class DemoClass {

  // creae a generics method
  public <T> void genericsMethod(T data) {
    System.out.println("Generics Method:");
    System.out.println("Data Passed: " + data);
  }
}
```

Generics

Bounded Types



```
public class Box<T> {
  private T value;
}
```

- Allows you to restrict the types that can be used as type arguments for a generic class, interface, or method.

- Ensures that the type parameters meet certain criteria which is useful for:
  - maintaining type safety
  - leveraging polymorphism

Types of Bounded Types :

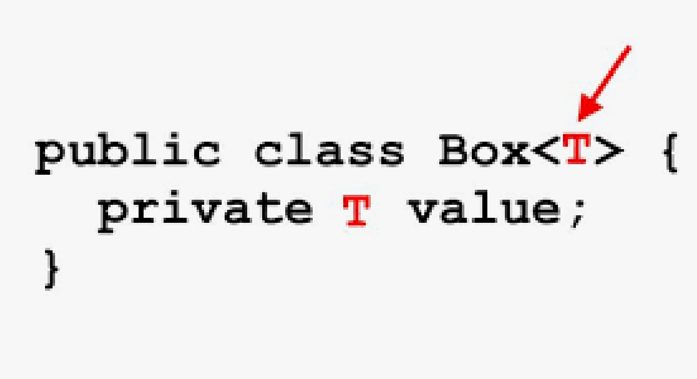
Upper Bounded Types (extends)
Lower Bounded Types (super)

Generics

Upper Bounded Types

An upper bounded type restricts the type parameter to be a specific type or a subtype of that type. This is specified using the extends keyword.

<T extends SomeClass>



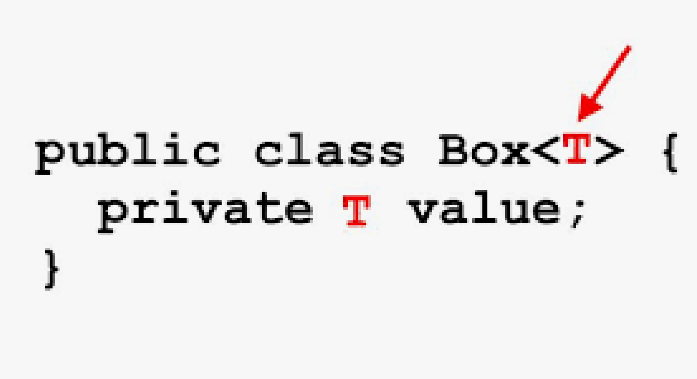
```
public class Box<T> {
  private T value;
}
```

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

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

public T get() {return t;}

public void print()
{System.out.println(t.doubleValue());}
}
```

Generics

Upper Bounded Types

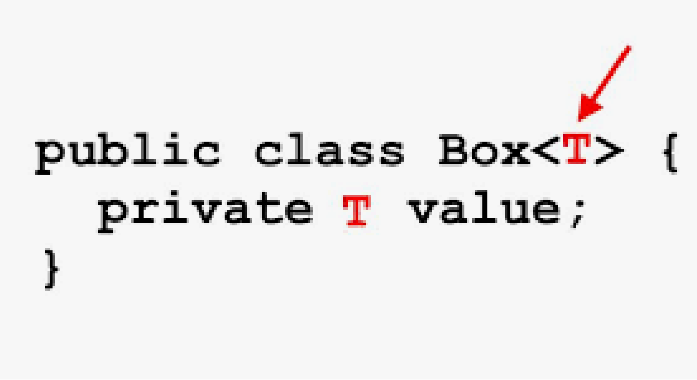
The Box class can only be instantiated with types that are subclasses of Number (e.g., Integer, Double), ensuring that the type has a doubleValue() method.



```
public class Box<T> {
   private T value;
}
```

```
public static void main(String[] args) {
Box<Integer> intBox = new Box<>();
intBox.set(10);
intBox.print(); // Output: 10.0

Box<Double> doubleBox = new Box<>();
doubleBox.set(10.5);
doubleBox.print(); // Output: 10.5

// Box<String> stringBox = new Box<>();
}
```

Generics



```
public class Box<T> {
   private T value;
}
```

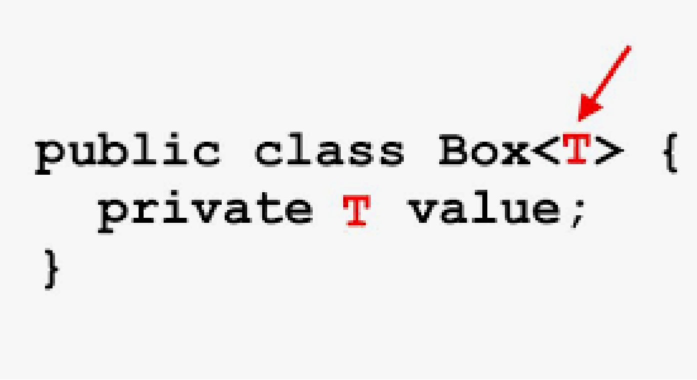
Lower Bounded Types

A lower bounded type restricts the type parameter to be a specific type or a supertype of that type. This is specified using the super keyword.

<T super SomeClass>

```
public class BoxPriter {
public static void addNumbers(List<? super Integer> list)
{
        for (int i = 1; i <= 10; i++) {
        list.add(i);
        }
}
```

Generics



```
public class Box<T> {
  private T value;
}
```

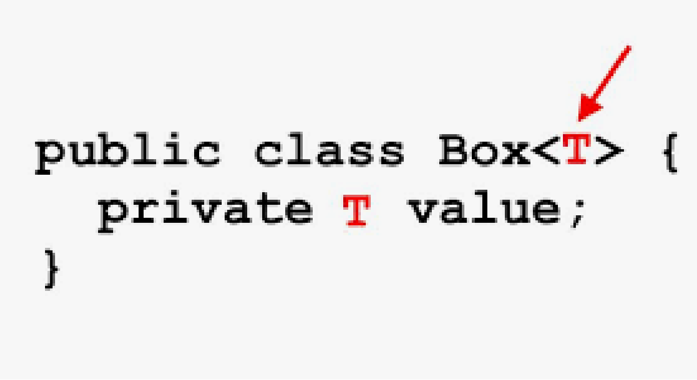
Lower Bounded Types

A lower bounded type restricts the type parameter to be a specific type or a supertype of that type. This is specified using the super keyword.

```
public static void main(String[] args) {
List<Number> numberList = new ArrayList<>();
addNumbers(numberList);
System.out.println(numberList); // Output:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

List<Object> objectList = new ArrayList<>();
addNumbers(objectList);
System.out.println(objectList); // Output:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
}
```

Generics

Wildcards



```
public class Box<T> {
   private T value;
}
```

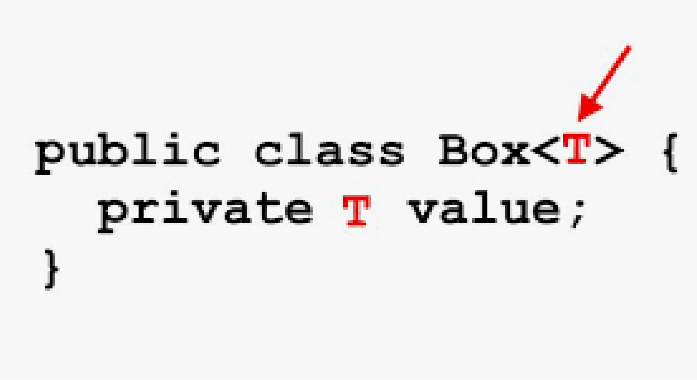
- Wildcards are special symbols used in generics to represent an unknown type.

- Wildcards allow for more flexible and reusable code by letting you specify a range of acceptable types for a generic class, interface, or method.

- There are three types of wildcards:

  unbounded wildcards
  upper bounded wildcards (bounded wildcards)
  lower bounded wildcards.

Generics



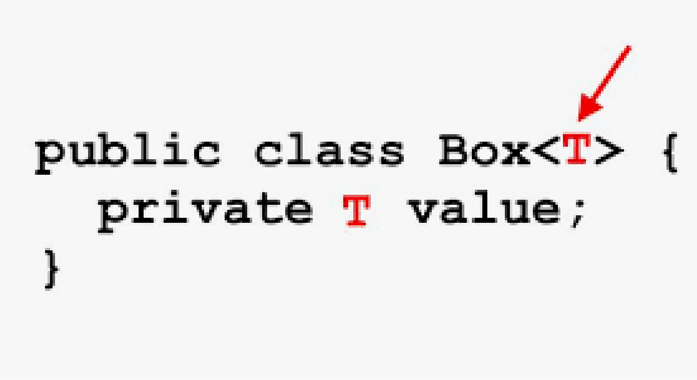
```
public class Box<T> {
  private T value;
}
```

Wildcards

Types of Wildcards

- Unbounded Wildcard (?)
- Upper Bounded Wildcard (? extends Type)
- Lower Bounded Wildcard (? super Type)

Generics



```
public class Box<T> {
  private T value;
}
```

Wildcards

Unbounded Wildcard (?)

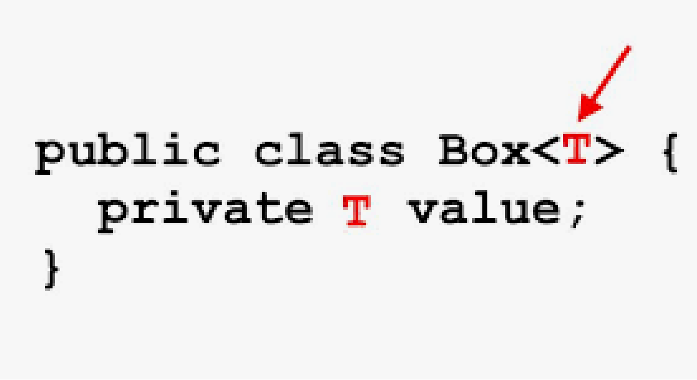
An unbounded wildcard represents an unknown type. It can be any type, similar to using Object, but it retains generic type information.

```
List<?> list = new
ArrayList<>();
```

Generics

Wildcards

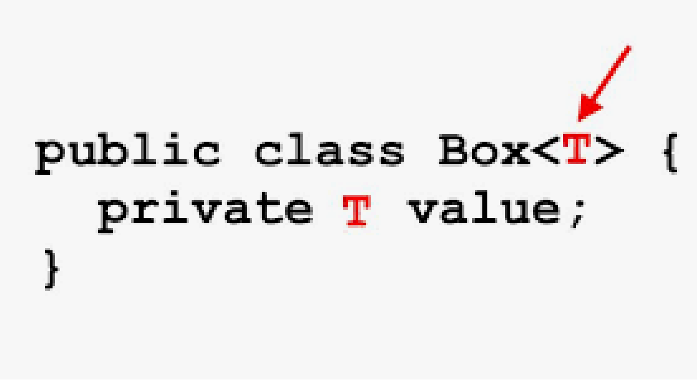
Unbounded Wildcard (?)



```
public class Box<T> {
  private T value;
}
```

```
public static void printList(List<?> list) {
for (Object elem : list) {System.out.println(elem);}}

public static void main(String[] args) {
List<String> stringList=List.of("apple","banana","cherry");
List<Integer> intList = List.of(1, 2, 3);
printList(stringList); // Output: apple, banana, cherry
printList(intList); // Output: 1, 2, 3
}
```

Generics



```
public class Box<T> {
  private T value;
}
```

Wildcards

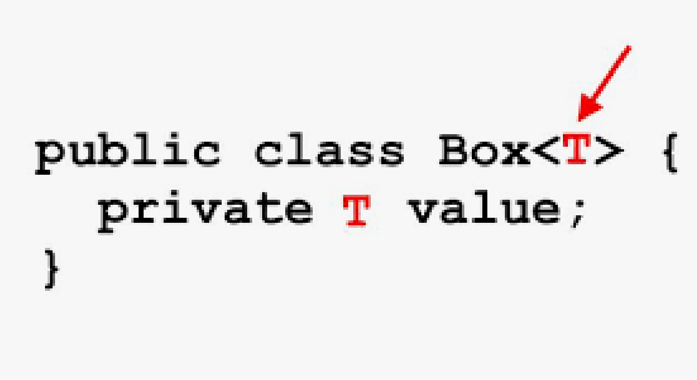
Upper Bounded Wildcard (? extends Type)

An upper bounded wildcard restricts the unknown type to be a specific type or a subtype of that type. This is specified using the extends keyword.

```
List<? extends Number> list = new
ArrayList<>();
```

Generics

Wildcards



```
public class Box<T> {
  private T value;
}
```

Upper Bounded Wildcard (? extends Type)

```
static double sumOfList(List<? extends Number> list) {
double sum = 0.0;
for (Number num : list) {sum += num.doubleValue();}
return sum;}

public static void main(String[] args) {
List<Integer> intList = List.of(1, 2, 3);
List<Double> doubleList = List.of(1.1, 2.2, 3.3);
System.out.println(sumOfList(intList)); // Output: 6.0
System.out.println(sumOfList(doubleList)); // Output: 6.6
}
```

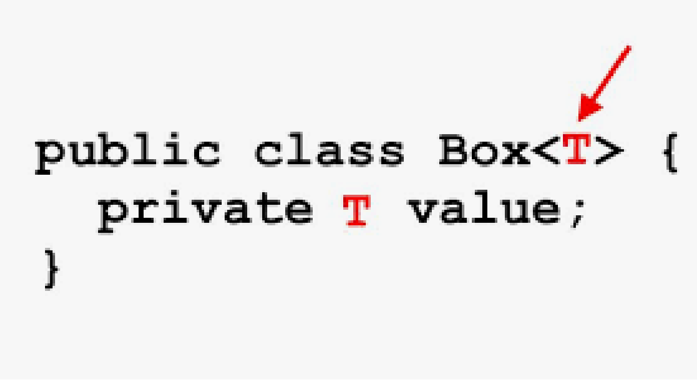
Generics

Wildcards

Lower Bounded Wildcard (? super Type)

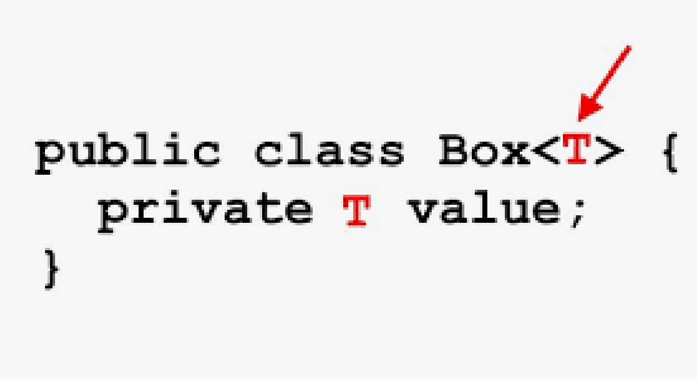
A lower bounded wildcard restricts the unknown type to be a specific type or a supertype of that type. This is specified using the super keyword.

```
List<? super Integer> list = new ArrayList<>();
```



```
public class Box<T> {
  private T value;
}
```

Generics



```
public class Box<T> {
   private T value;
}
```

Wildcards

Lower Bounded Wildcard (? super Type)

```
public static void addNumbers(List<? super Integer> list)
{for (int i = 1; i <= 10; i++) {list.add(i);}}

public static void main(String[] args) {
List<Number> numberList = new ArrayList<>();
List<Object> objectList = new ArrayList<>();
addNumbers(numberList);
addNumbers(objectList);
System.out.println(numberList); // Output: [1, 2, 3, 4, 5,
6, 7, 8, 9, 10]
System.out.println(objectList); // Output: [1, 2, 3, 4, 5,
6, 7, 8, 9, 10]
}
```

Generics

Thank You !!