Revisiting the Account Class

- Earlier we discussed the number field in the Account class

```
public class Account {
  private int number;
  ...
```

Revisiting the Account Class

- Earlier we discussed the number field in the Account class

```
public class Account {
  private int number;
  ...
```

Revisiting the Account Class

- Earlier we discussed the number field in the Account class

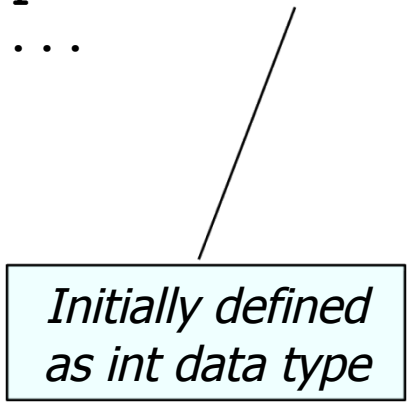
```
public class Account {
  private int number;
  ...
```

See previous lesson on "New Objects in Old Places"

Revisiting the Account Class

- Earlier we discussed the number field in the Account class

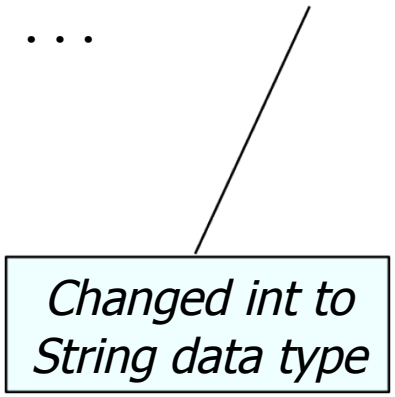
```
public class Account {
  private int number;
  ...
```



Revisiting the Account Class

- Earlier we discussed the number field in the Account class

```
public class Account {
  private String number;
  ...
```



How does this change affect reuse?

Revisiting the Account Class

- Should we create two classes differing only by type of number?

```
public class AccountInt {
  private int number;
  ...
```

```
public class AccountString {
  private String number;
  ...
```

Revisiting the Account Class

- Should we create two classes differing only by type of number?

```
public class AccountInt {
  private int number;
  ...
```

```
public class AccountString {
  private String number;
  ...
```

This approach is not scalable since multiple versions are harder to support

Revisiting the Account Class

- Should we include two fields & only use one for an implementation?

```
public class AccountInt {
   private int intNum;
  private String stringNum;
  ...
```

Revisiting the Account Class

- Should we include two fields & only use one for an implementation?

```
public class AccountInt {
  private int intNum;
  private String stringNum;
  ...
```

This approach leads to errors & confusion later

Overview of Generic Programming

- Java supports a variant of *generic programming*

See en.wikipedia.org/wiki/Generic_programming

Overview of Generic Programming

- Java supports a variant of *generic programming*
  - Algorithms & data structures can be written in terms of "placeholder" types

See en.wikipedia.org/wiki/Generic_programming

Overview of Generic Programming

- Java supports a variant of *generic programming*
  - Algorithms & data structures can be written in terms of "placeholder" types
  - Placeholders are later filled in when concrete types are provided as parameters to objects

See en.wikipedia.org/wiki/Generic_programming

Overview of Java Generics

- A Java class can be parameterized

```
public class SomeClass<T> {
  private T aField;

  public SomeClass(T param)
  { aField = param; }

  public T getField()
  { return aField; }
  ...
```

See en.wikipedia.org/wiki/Generics_in_Java

Overview of Java Generics

- A Java class can be parameterized
  - Enables passing & returning of ADTs as parameters to a class & its methods

```
public class SomeClass<T> {
  private T aField;

  public SomeClass(T param)
  { aField = param; }

  public T getField()
  { return aField; }
  ...
```

See en.wikipedia.org/wiki/Generics_in_Java

Overview of Java Generics

- A Java class can be parameterized
  - Enables passing & returning of ADTs as parameters to a class & its methods
  - Class methods can be written once

```
public class SomeClass<T> {
  private T aField;

  public SomeClass(T param)
  { aField = param; }

  public T getField()
  { return aField; }
  ...
```

See en.wikipedia.org/wiki/Generics_in_Java

Overview of Java Generics

- A Java class can be parameterized
  - Enables passing & returning of ADTs as parameters to a class & its methods
  - Class methods can be written once
  - Methods are able to operate on objects of various types

```
public class SomeClass<T> {
  private T aField;

  public SomeClass(T param)
  { aField = param; }

  public T getField()
  { return aField; }
  ...
```

```
SomeClass<String> s1 =
  new SomeClass<>("hello world");
String s = s1.getField();
```

```
SomeClass<Integer> s2 =
  new SomeClass<>(42);
Integer I = s2.getField();
```

See en.wikipedia.org/wiki/Generics_in_Java

Overview of Java Generics

- Account can be a generic class parameterized by the type of the number field

```
public class Account<Number> {
  private Number number;
  ...

Account<String> a1 =
  new Account<>("Lucy Lastic",
                "098453");

Account<Integer> a2 =
  new Account<>("Sue Flockey",
                853405);
```

Overview of Java Generics

- Objects of generic Account class can be instantiated with desired number type

```
public class Account<Number> {
  private Number number;
  ...

Account<String> a1 =
  new Account<>("Lucy Lastic",
                "098453");

Account<Integer> a2 =
  new Account<>("Sue Flockey",
                853405);
```

Overview of Java Generics

- Objects of generic Account class can be instantiated with desired number type

```
public class Account<Number> {
   private Number number;
   ...

Account<String> a1 =
  new Account<>("Lucy Lastic",
                "098453");

Account<Integer> a2 =
  new Account<>("Sue Flockey",
                853405);
```

In Java 7 (& beyond) the compiler can deduce the type arguments to new

Overview of Java Generics

- A generic class in Java is defined using the syntax shown here

```
class name<T1, T2, ..., Tn> {
  // ...
}
```

Overview of Java Generics

- A generic class in Java is defined using the syntax shown here

```
class name<T1, T2, ..., Tn> {
  // ...
}
```

The type parameter section is enclosed in angle brackets

Overview of Java Generics

- A generic class in Java is defined using the syntax shown here

```
class name<T1, T2, ..., Tn> {
  // ...
}
```

The type parameter section is enclosed in angle brackets

Overview of Java Generics

- A generic class in Java is defined using the syntax shown here

```
class name<T1, T2, ..., Tn> {
  // ...
}
```

The type parameter section is enclosed in angle brackets

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account {
  private String number;
  ...
  public String getNumber(){
    return number;
  }
}
```

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account {
   private String number;
   ...
   public String getNumber(){
      return number;
   }
}
```

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account<Number> {
  private Number number;
  ...
  public Number getNumber(){
    return number;
  }
}
```

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account<Number> {
  private Number number;
  ...
  public Number getNumber(){
    return number;
  }
}
```

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account<Number> {
  private Number number;
  ...
  public Account(String name,
                 Number num) {
    number = num;
    ...
  }

  public Number getNumber(){
    return number;
  }
}
```

Overview of Java Generics

- It's easy to update the Account class to use generics

```
public class Account<Number> {
  private Number number;
  ...
  public Account(String name,
                 Number num) {
    number = num;
    ...
  }

  public Number getNumber(){
    return number;
  }
}
```

Overview of Java Generics

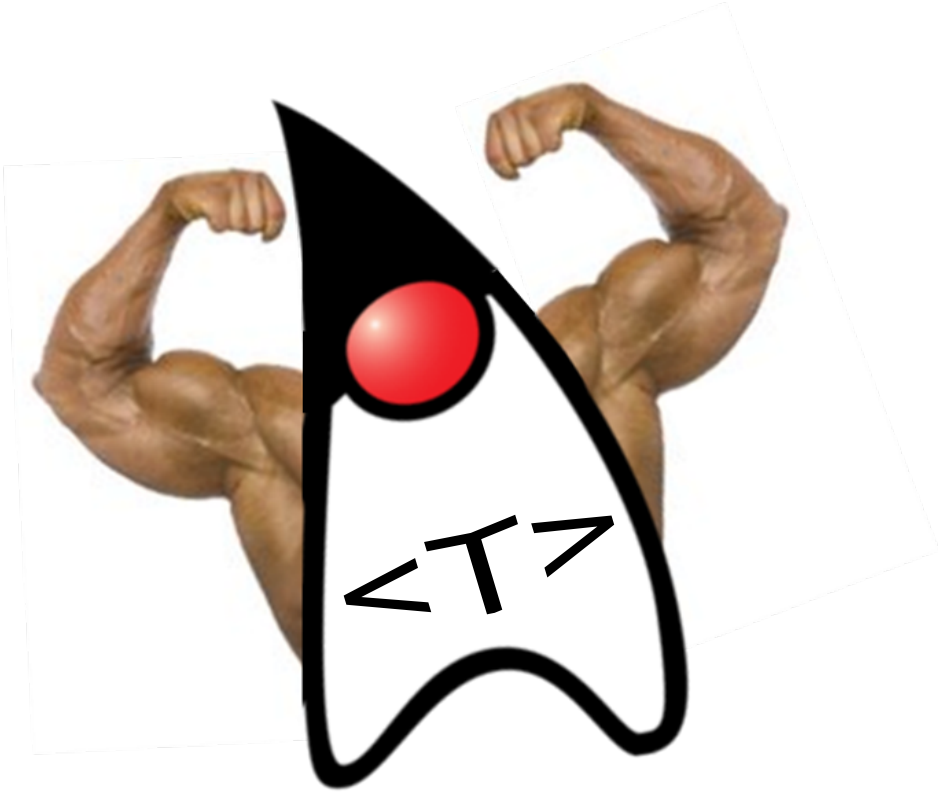
- It's easy to update the Account class to use generics

```
public class Account<Number> {
  private Number number;
  ...
  public Account(String name,
                 Number num) {
    number = num;
    ...
  }

  public Number getNumber(){
    return number;
  }
}
```

Benefits of Java Generics

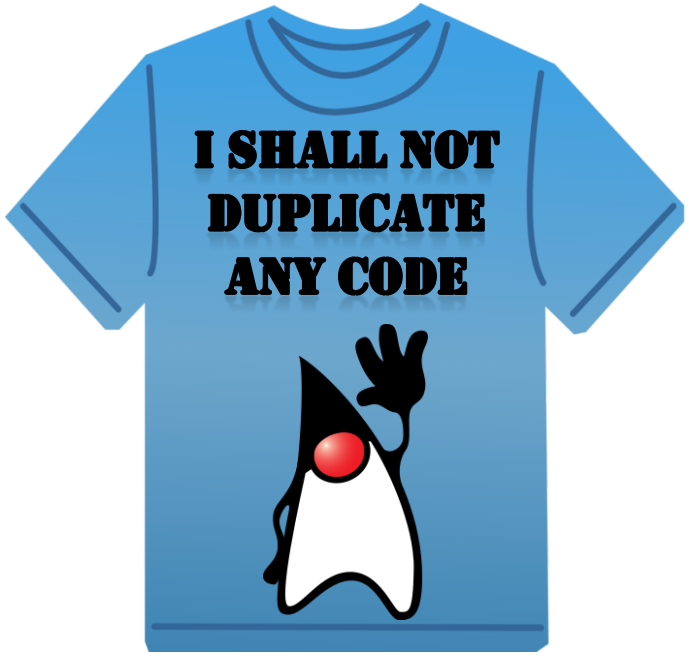
- Java generics offer several benefits



See docs.oracle.com/javase/tutorial/extra/generics

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication




```
public class Account<Number> {
  private Number number;
  ...
```

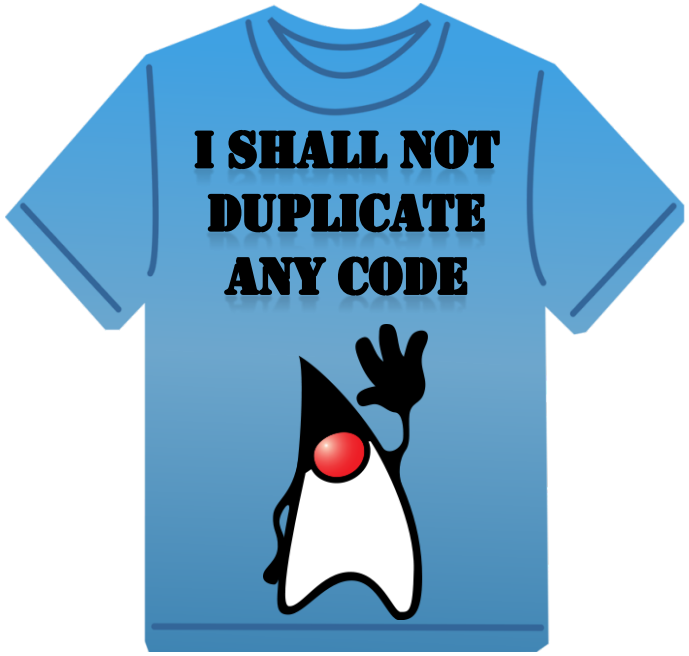
```
Account<String> a1 =
  new Account<>(...);

Account<Integer> a2 =
  new Account<>(...);
```

See en.wikipedia.org/wiki/Don't_repeat_yourself

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication



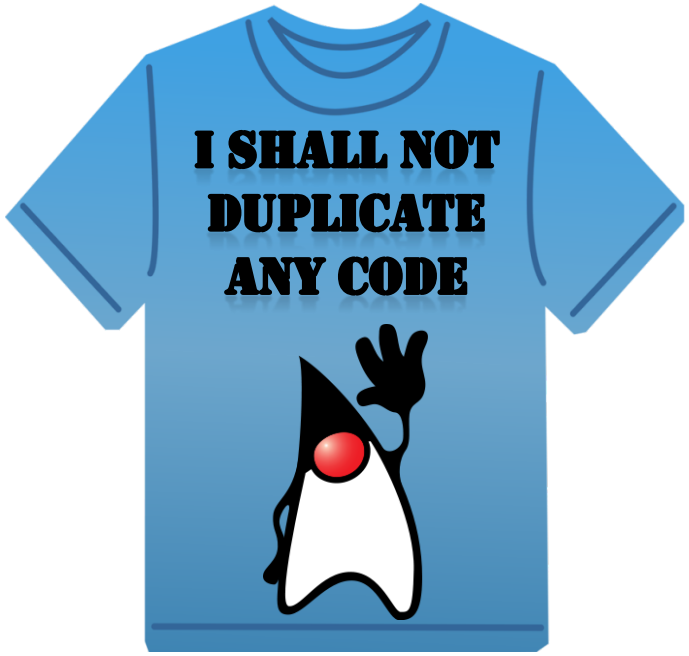

```
public class Account<Number> {
  private Number number;
  ...
```

```
Account<String> a1 =
  new Account<>(...);

Account<Integer> a2 =
  new Account<>(...);
```

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication




```
public class Account<Number> {
  private Number number;
  ...
```

```
Account<String> a1 =
  new Account<>(...);

Account<Integer> a2 =
  new Account<>(...);
```

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication
  - Ensure compile-time type safety

```
public class Account<Number> {
  private Number number;
  ...


Account<String> a1 =
  new Account<>("Bob", 674903);

Account<Integer> a2 =
  new Account<>("Bob", "674903
  ");
```

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication
  - Ensure compile-time type safety

```
public class Account<Number> {
  private Number number;
  ...


Account<String> a1 = // Fails!
  new Account<>("Bob", 674903);

Account<Integer> a2 =
  new Account<>("Bob", "674903
  ");
```

Benefits of Java Generics

- Java generics offer several benefits
  - Reduce unnecessary code duplication
  - Ensure compile-time type safety

```
public class Account<Number> {
  private Number number;
  ...


Account<String> a1 =
  new Account<>("Bob", 674903);

Account<Integer> a2 = // Fails!
  new Account<>("Bob", "674903
  ");
```

Restrictions on Using Java Generics

- There are some restrictions on using Java generics

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types

```
public class Account<Number> {
  private Number number;
  ...


Account<int> a1 = // Fails!
  new Account<>("Bob", 674903);

Account<Integer> a2 = // Works!
  new Account<>("Bob", 674903);
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#instantiate

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types

```
public class Account<Number> {
  private Number number;
  ...


Account<int> a1 = // Fails!
  new Account<>("Bob", 674903);

Account<Integer> a2 = // Works!
  new Account<>("Bob", 674903);
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#instantiate

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types

```
public class Account<Number> {
  private Number number;
  ...


Account<int> a1 = // Fails!
  new Account<>("Bob", 674903);

Account<Integer> a2 = // Works!
  new Account<>("Bob", 674903);
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#instantiate

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types
  - Cannot create new instances of type parameters

```
public class Account<Number> {
  private Number number;
  private Number defaultNumber;
  ...

  public Account(.....) {
    // Fails!
    defaultNumber = new Number();
  }
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createObjects

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types
  - Cannot create new instances of type parameters

```
public class Account<Number> {
  private Number number;
  private Number defaultNumber;
  ...

  public Account(.....) {
    // Fails!
    defaultNumber = new Number();
  }
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createObjects

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types
  - Cannot create new instances of type parameters

```
public class Account<Number> {
  private Number number;
  private Number defaultNumber;
  ...

  public Account(.....
                 Class<Number> c) {
    // Works!
    defaultNumber =
      c.newInstance()();
  }
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createObjects

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types
  - Cannot create new instances of type parameters
  - Cannot declare static fields whose types are type parameters

```
public class Account<Number> {
  private static Number
    defaultAccountNumber; // Fails!
  ...
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createStatic

Restrictions on Using Java Generics

- There are some restrictions on using Java generics, e.g.,
  - Cannot instantiate generic types of primitive types
  - Cannot create new instances of type parameters
  - Cannot declare static fields whose types are type parameters

```
public class Account<Number> {
  private static Number
    defaultAccountNumber; // Fails!
  ...
```

See docs.oracle.com/javase/tutorial/java/generics/restrictions.html#createStatic

Restrictions on Using Java Generics

- There restrictions on Java generics largely stem from its "type erasure" semantics

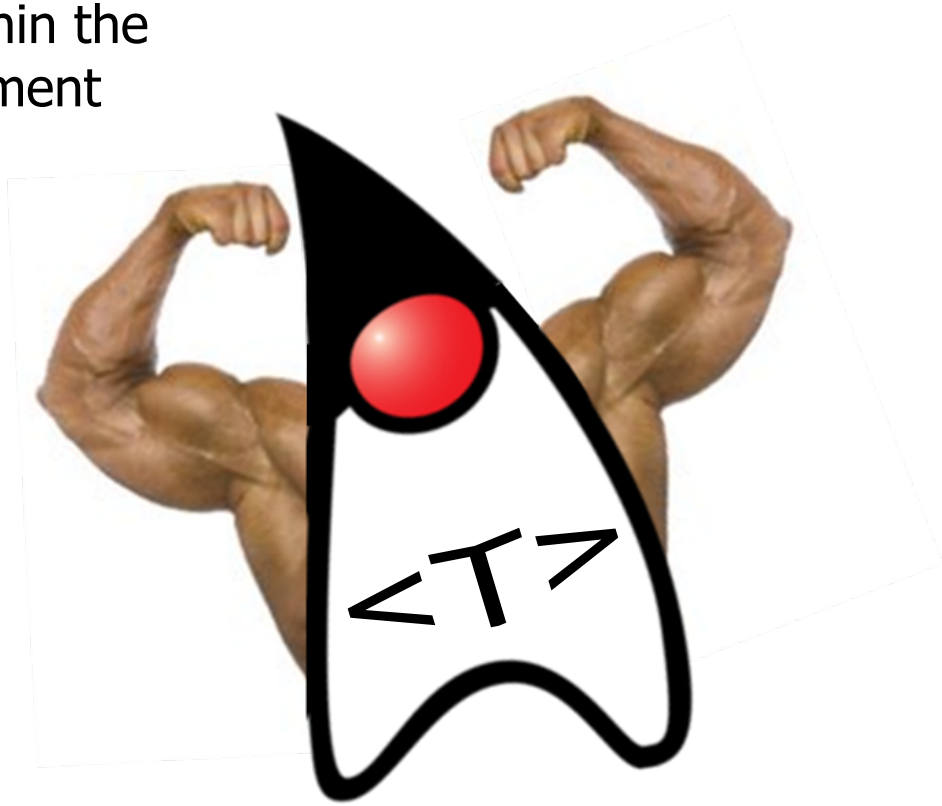
See code.stephenmorley.org/articles/java-generics-type-erasure

Conclusion

- When there are multiple ways to represent key properties of an ADT consider using Java generics

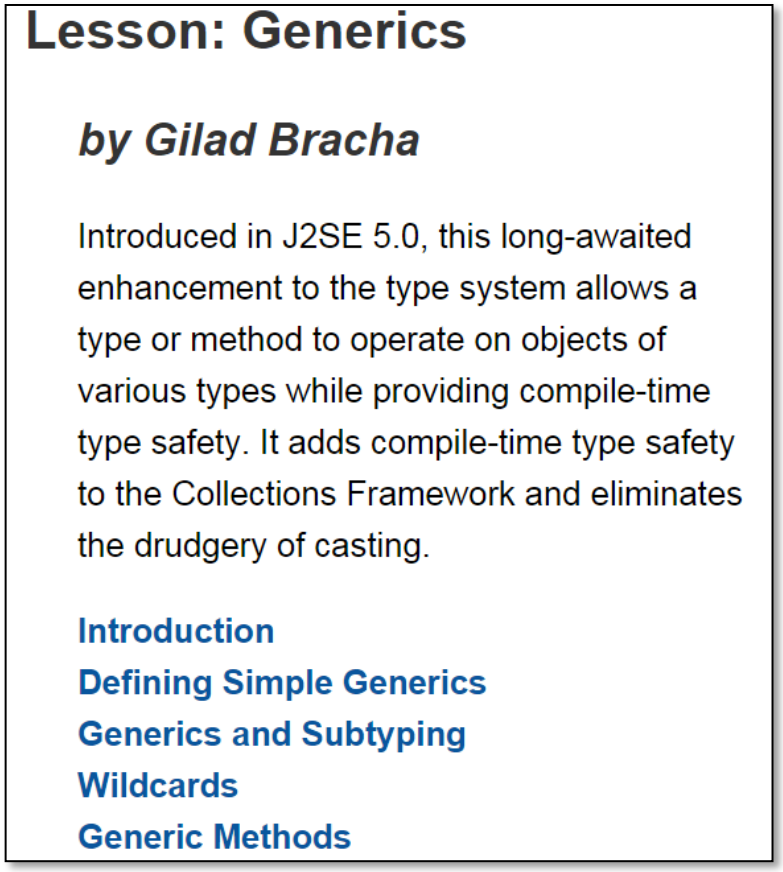
Conclusion

- When generics are used properly within the constraints of Java's runtime environment they provide powerful capabilities
  - e.g., stronger type checking at compile-time & systematic reuse via generic programming



Conclusion

- More information on Java generics is available in the following tutorials:
  - docs.oracle.com/javase/tutorial/java/generics
  - docs.oracle.com/javase/tutorial/extra/generics



Lesson: Generics

by Gilad Bracha

Introduced in J2SE 5.0, this long-awaited enhancement to the type system allows a type or method to operate on objects of various types while providing compile-time type safety. It adds compile-time type safety to the Collections Framework and eliminates the drudgery of casting.

Introduction
Defining Simple Generics
Generics and Subtyping
Wildcards
Generic Methods