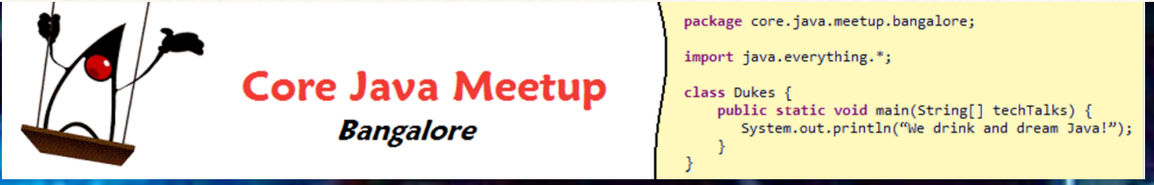



```
package core.java.meetup.bangalore;

import java.everything.*;

class Dukes {
    public static void main(String[] techTalks) {
       System.out.println("We drink and dream Java!");
    }
}
```


JAVA GENERICS: BY EXAMPLE

Ganesh Samarthyam

ganesh@codeops.tech


THE QUESTION


TEA OR COFFEE?
MORNING?
N
Y
YAWNING?
N
Y
TEA
COFFEE


while (sleepy){
fillCup();
drinkCoffee();
}

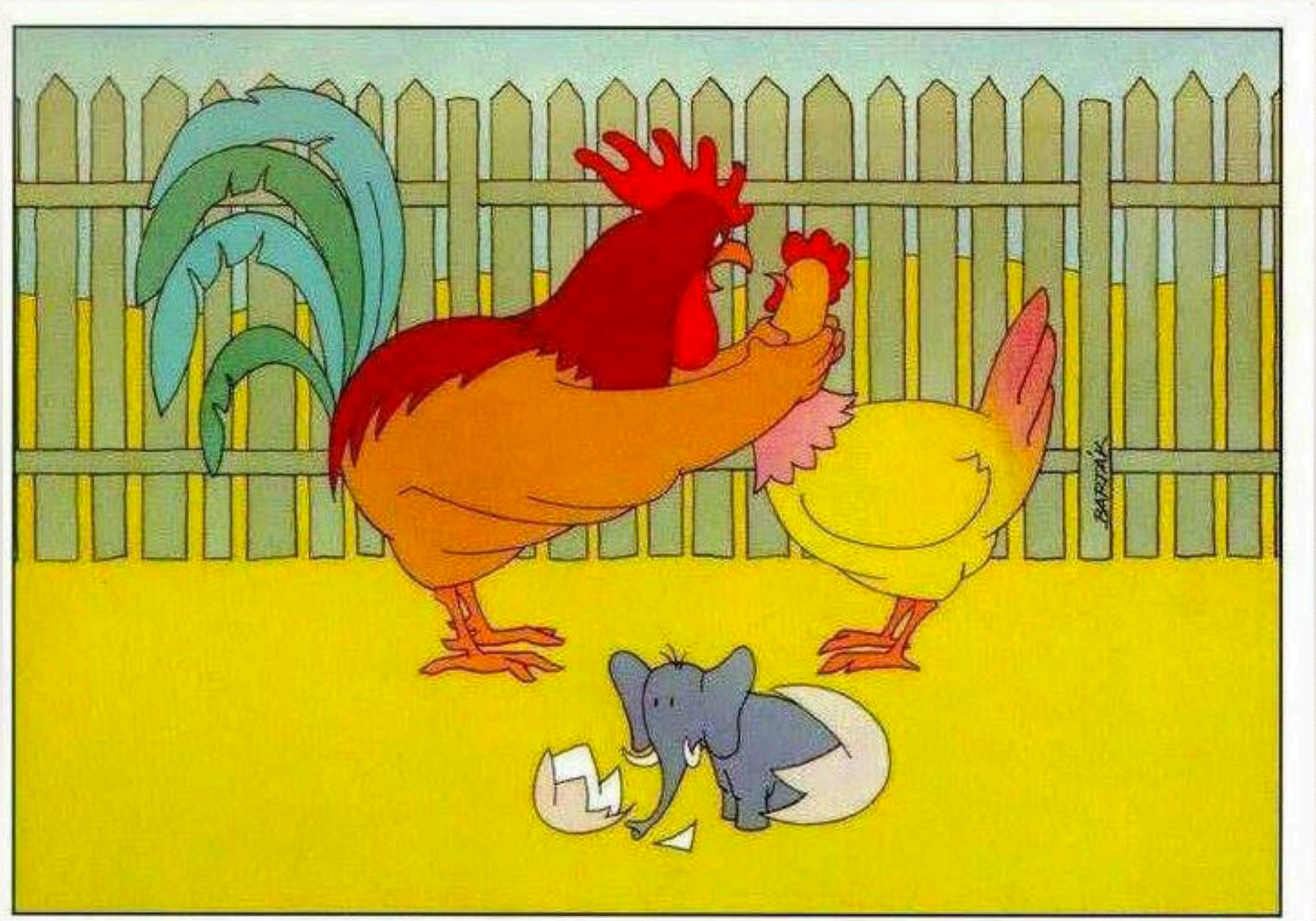

Cup<T>

Why should I use generics?

Avoid repetition

Reduce ctrl+c and ctrl+v

Generics: Type safety



Source code

https://github.com/CodeOpsTech/JavaGenerics

First program

```
// This program shows container implementation using generics
class BoxPrinter<T> {
    private T val;
    public BoxPrinter(T arg) {
        val = arg;
    }
    public String toString() {
        return "[" + val + "]";
    }
}

class BoxPrinterTest {
   public static void main(String []args) {
        BoxPrinter<Integer> value1 = new BoxPrinter<Integer>(new Integer(10));
        System.out.println(value1);

        BoxPrinter<String> value2 = new BoxPrinter<String>("Hello world");
        System.out.println(value2);
   }
}
```

```
[10]
[Hello world]
```

Another example

```
// It demonstrates the usage of generics in defining classes
class Pair<T1, T2> {
    T1 object1;
    T2 object2;
    Pair(T1 one, T2 two) {
            object1 = one;
            object2 = two;
    }
    public T1 getFirst() {
            return object1;
    }
    public T2 getSecond() {
            return object2;
    }
}

class PairTest {
    public static void main(String []args) {
        Pair<Integer, String> worldCup = new Pair<Integer, String>(2018, "Russia");
        System.out.println("World cup " +  worldCup.getFirst() +
            " in " + worldCup.getSecond());
    }
}
```

```
World cup 2018 in Russia
```

Type checking

```
Pair<Integer, String> worldCup = new Pair<String, String>(2018, "Russia");
```

```
TestPair.java:20: cannot find symbol
symbol : constructor Pair(int,java.lang.String)
location: class Pair<java.lang.String,java.lang.String>
```

Type checking

Pair<Integer, String> worldCup = new Pair<Number, String>(2018, “Russia");

TestPair.java:20: incompatible types
found : Pair<java.lang.Number,java.lang.String>
required: Pair<java.lang.Integer,java.lang.String>

Yet another example

```
// This program shows how to use generics in your programs
class PairOfT<T> {
    T object1;
    T object2;
    PairOfT(T one, T two) {
        object1 = one;
        object2 = two;
    }
    public T getFirst() {
        return object1;
    }
    public T getSecond() {
        return object2;
    }
}
```

Type checking

```
PairOfT<Integer, String> worldCup = new PairOfT<Integer, String>(2018, "Russia");
```

Type checking

```
PairOfT<String> worldCup = new PairOfT<String>(2018, “Russia");
```

```
TestPair.java:20: cannot find symbol
symbol : constructor PairOfT(int,java.lang.String)
location: class PairOfT<java.lang.String>
PairOfT<String> worldCup = new PairOfT<String>(2018, "Russia");
```

```
PairOfT<String> worldCup = new PairOfT<String>("2018", "Russia");
```

Java => Too much typing

Diamond syntax

```
// This program shows the usage of the diamond syntax when using generics
class Pair<T1, T2> {
    T1 object1;
    T2 object2;

    Pair(T1 one, T2 two) {
        object1 = one;
        object2 = two;
    }

    public T1 getFirst() {
        return object1;
    }

    public T2 getSecond() {
        return object2;
    }
}

class TestPair {
    public static void main(String []args) {
        Pair<Integer, String> worldCup = new Pair<>(2018, "Russia");
        System.out.println("World cup " +  worldCup.getFirst() + " in " + worldCup.getSecond());
    }
}
```

World cup 2018 in Russia

Forgetting < and >

```
Pair<Integer, String> worldCup = new Pair(2018, "Russia");
```

```
Pair.java:19: warning: [unchecked] unchecked call to Pair(T1,T2) as a member of the
raw type Pair
Pair<Integer, String> worldCup = new Pair(2018, "Russia");
                                         ^
where T1,T2 are type-variables:
T1 extends Object declared in class Pair
T2 extends Object declared in class Pair

Pair.java:19: warning: [unchecked] unchecked conversion
Pair<Integer, String> worldCup = new Pair(2018, "Russia");
                                         ^
required: Pair<Integer,String>
found: Pair
2 warnings
```

Type erasure

Implementation of generics is static in nature, which means that the Java compiler interprets the generics specified in the source code and replaces the generic code with concrete types. This is referred to as **type erasure**. After compilation, the code looks similar to what a developer would have written with concrete types.

Type erasure

Type erasure

T mem = new T(); // wrong usage - compiler error

Type erasure

T[] amem = new T[100]; // wrong usage - compiler error

Type erasure

```
class X<T> {
    T instanceMem; // okay
    static T statMem; // wrong usage - compiler error
}
```

Generics - limitations

You cannot instantiate a generic type with primitive types, for example, List<int> is not allowed. However, you can use boxed primitive types like List<Integer>.

Raw types

```
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;

class RawTest1 {
    public static void main(String []args) {
        List list = new LinkedList();
      list.add("First");
      list.add("Second");
      List<String> strList = list;  //#1
      for(Iterator<String> itemItr = strList.iterator(); itemItr.hasNext();)
          System.out.println("Item: " + itemItr.next());

   List<String> strList2 = new LinkedList<>();
   strList2.add("First");
   strList2.add("Second");
   List list2 = strList2;   //#2
   for(Iterator<String> itemItr = list2.iterator(); itemItr.hasNext();)
          System.out.println("Item: " + itemItr.next());
    }
}
```

```
Item: First
Item: Second
Item: First
Item: Second
```

```
$ javac RawTest1.java
Note: RawTest1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

Raw types

```
$ javac -Xlint:unchecked RawTest1.java
RawTest1.java:14: warning: [unchecked] unchecked call to add(E) as a member of the raw type List
      list.add("First");
               ^
  where E is a type-variable:
    E extends Object declared in interface List
RawTest1.java:15: warning: [unchecked] unchecked call to add(E) as a member of the raw type List
      list.add("Second");
               ^
  where E is a type-variable:
    E extends Object declared in interface List
RawTest1.java:16: warning: [unchecked] unchecked conversion
      List<String> strList = list;  //#1
                             ^
  required: List<String>
  found:    List
RawTest1.java:24: warning: [unchecked] unchecked conversion
    for(Iterator<String> itemItr = list2.iterator(); itemItr.hasNext();)
                                                     ^
  required: Iterator<String>
  found:    Iterator
4 warnings
$
```

Raw types

```
import java.util.List;
import java.util.LinkedList;
import java.util.Iterator;

class RawTest2 {
    public static void main(String []args) {
        List list = new LinkedList();
        list.add("First");
        list.add("Second");
        List<String> strList = list;
        strList.add(10); // #1: generates compiler error
        for(Iterator<String> itemItr = strList.iterator(); itemItr.hasNext();)
            System.out.println("Item : " + itemItr.next());

        List<String> strList2 = new LinkedList<>();
        strList2.add("First");
        strList2.add("Second");
        List list2 = strList2;
        list2.add(10); // #2: compiles fine, results in runtime exception
        for(Iterator<String> itemItr = list2.iterator(); itemItr.hasNext();)
            System.out.println("Item : " + itemItr.next());
    }
}
```

Raw types

```
List<String> strList2 = new LinkedList<>();
strList2.add("First");
strList2.add("Second");
List list2 = strList2;
list2.add(10); // #2: compiles fine, results in runtime exception
for(Iterator<String> itemItr = list2.iterator(); itemItr.hasNext();)
        System.out.println("Item : " + itemItr.next());
```

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot
be cast to java.lang.String at RawTest2.main(RawTest2.java:27)
```

Best practice

Avoid using raw types of generic types

Generic methods

```
// This program demonstrates generic methods
import java.util.List;
import java.util.ArrayList;

class Utilities {
    public static <T> void fill(List<T> list, T val) {
        for(int i = 0; i < list.size(); i++)
            list.set(i, val);
    }
}

class UtilitiesTest {
    public static void main(String []args) {
        List<Integer> intList = new ArrayList<Integer>();
        intList.add(10);
        intList.add(20);
        System.out.println("The original list is: " + intList);
        Utilities.fill(intList, 100);
        System.out.println("The list after calling Utilities.fill() is: " + intList);
    }
}
```

```
The original list is: [10, 20]
The list after calling Utilities.fill() is: [100, 100]
```

Generics and sub typing

Subtyping works for class types: you can assign a derived type object to its base type reference.

However, subtyping does not work for generic type parameters: you cannot assign a derived generic type parameter to a base type parameter.

Generics and sub typing

```
// illegal code – assume that the following intialization is allowed
List<Number> intList = new ArrayList<Integer>();
intList.add(new Integer(10)); // okay
intList.add(new Float(10.0f)); // oops!
```

Generics and sub typing

```
List<Number> intList = new ArrayList<Integer>();
```

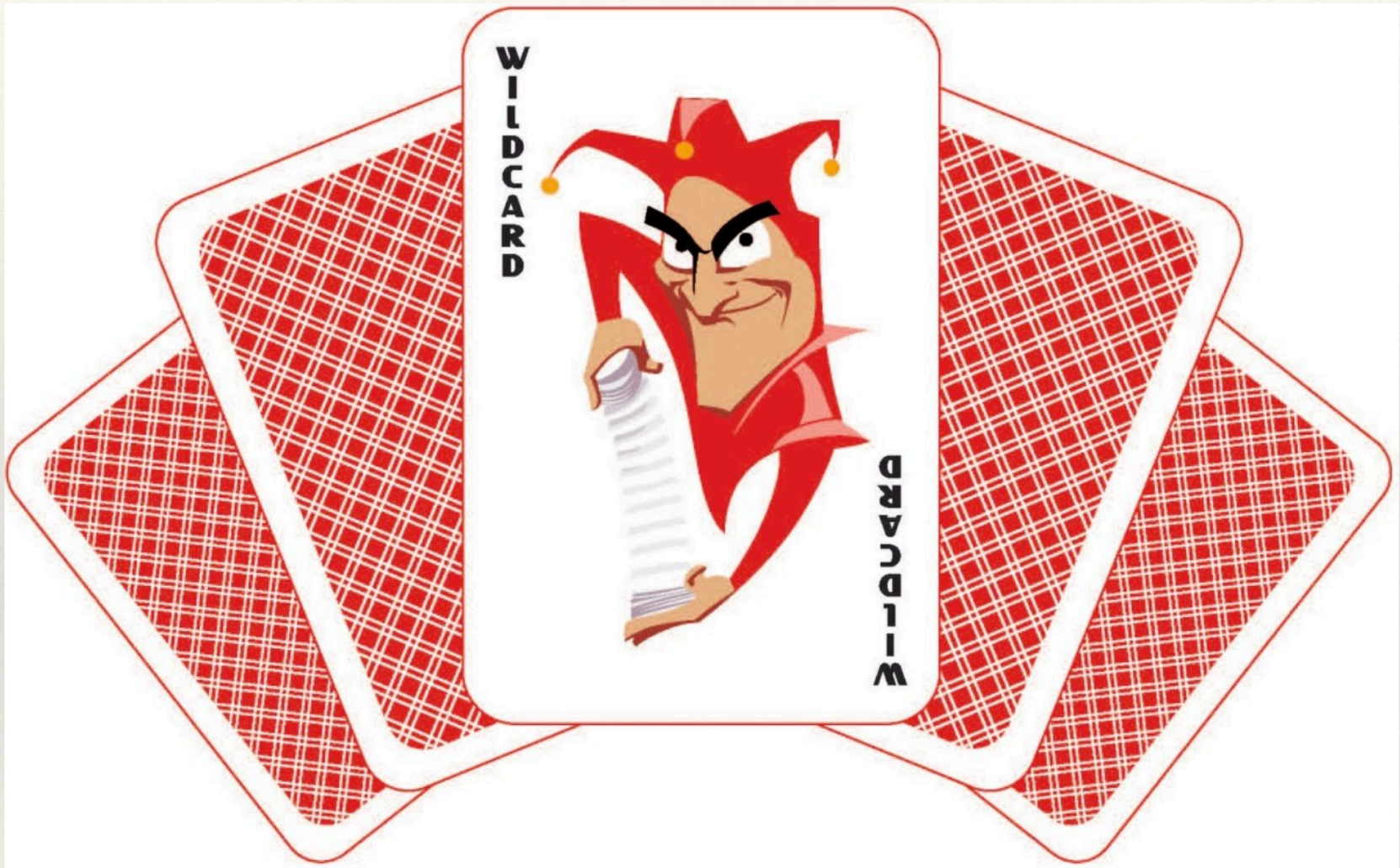
```
WildCardUse.java:6: incompatible types
found : java.util.ArrayList<java.lang.Integer>
required: java.util.List<java.lang.Number>
```

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

Wildcard types

Type parameters for generics have a limitation: generic type parameters should match exactly for assignments. To overcome this subtyping problem, you can use wildcard types.

Wildcard types



Use a wildcard to indicate that it can match for any type: List<?>, you mean that it is a List of any type—in other words, you can say it is a “list of unknowns!

How about this “workaround”?

List<Object> numList = new ArrayList<Integer>();

WildCardUse.java:6: incompatible types
found : java.util.ArrayList<java.lang.Integer>
required: java.util.List<java.lang.Object>
List<Object> numList = new ArrayList<Integer>();

Wildcard use

```
// This program demonstrates the usage of wild card parameters
import java.util.List;
import java.util.ArrayList;

class WildCardUse {
    static void printList(List<?> list){
        for(Object element: list)
            System.out.println("[" + element + "]");
    }

    public static void main(String []args) {
        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(100);
        printList(list);
        List<String> strList = new ArrayList<>();
        strList.add("10");
        strList.add("100");
        printList(strList);
    }
}
```

```
[10]
[100]
[10]
[100]
```

Limitations of Wildcards

```
List<?> wildCardList = new ArrayList<Integer>();
wildCardList.add(new Integer(10));
System.out.println(wildCardList);
```

```
WildCardUse.java:7: cannot find symbol
symbol : method add(java.lang.Integer)
location: interface java.util.List<capture#145 of ? extends java.lang.Number>
wildCardList.add(new Integer(10));
```

Wildcard types

In general, when you use wildcard parameters, you ***cannot*** call methods that ***modify*** the object. If youtry to modify, the compiler will give you confusing error messages. However, you ***can*** call methods that ***access*** the object.

More about generics

- It's possible to define or declare generic methods in an interface or a class even if the class or the interface itself is not generic.

- A generic class used without type arguments is known as a raw type. Of course, raw types are not type safe. Java supports raw types so that it is possible to use the generic type in code that is older than Java 5 (note that generics were introduced in Java 5). The compiler generates a warning when you use raw types in your code. You may use @SuppressWarnings({ "unchecked" }) to suppress the warning associated with raw types.

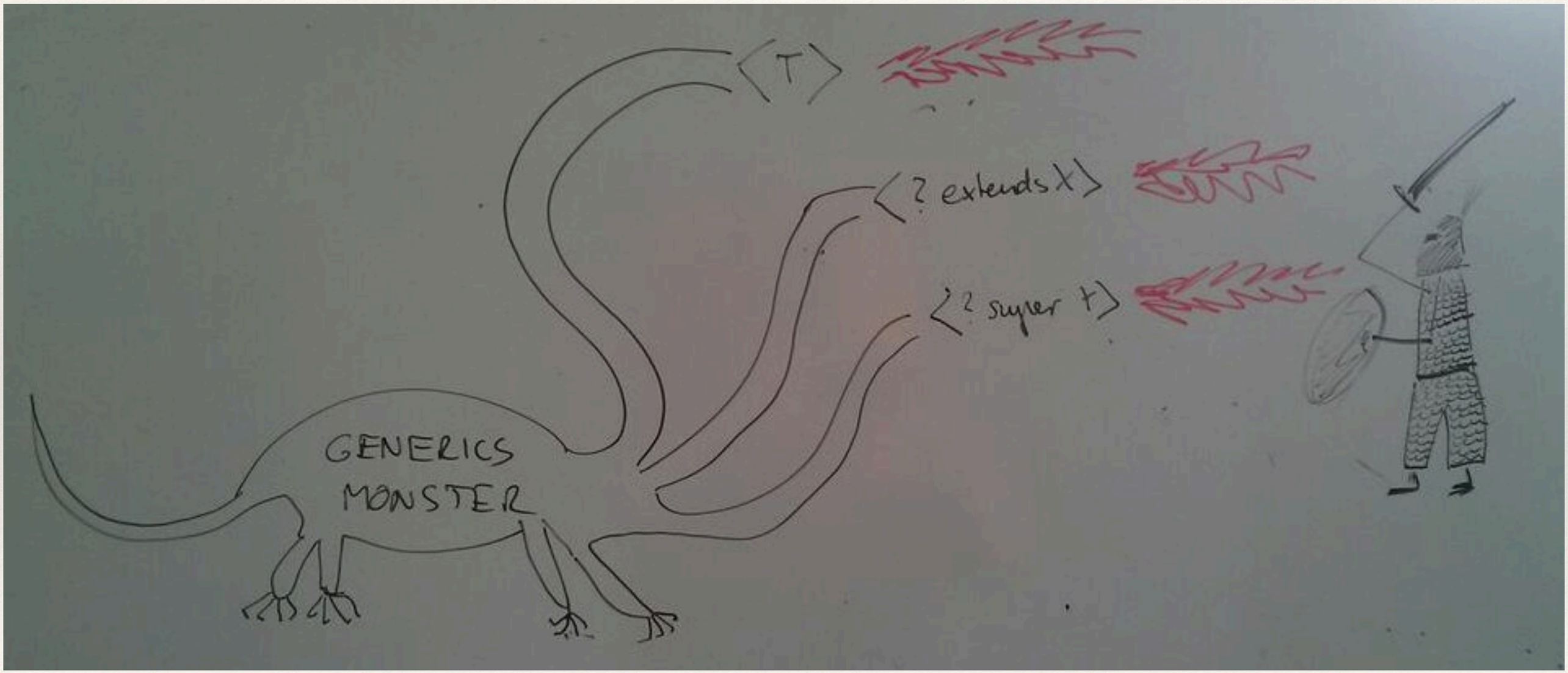
- List<?> is a supertype of any List type, which means you can pass List<Integer>, or List<String>, or even List<Object> where List<?> is expected.

<? super T> and <? extends T>

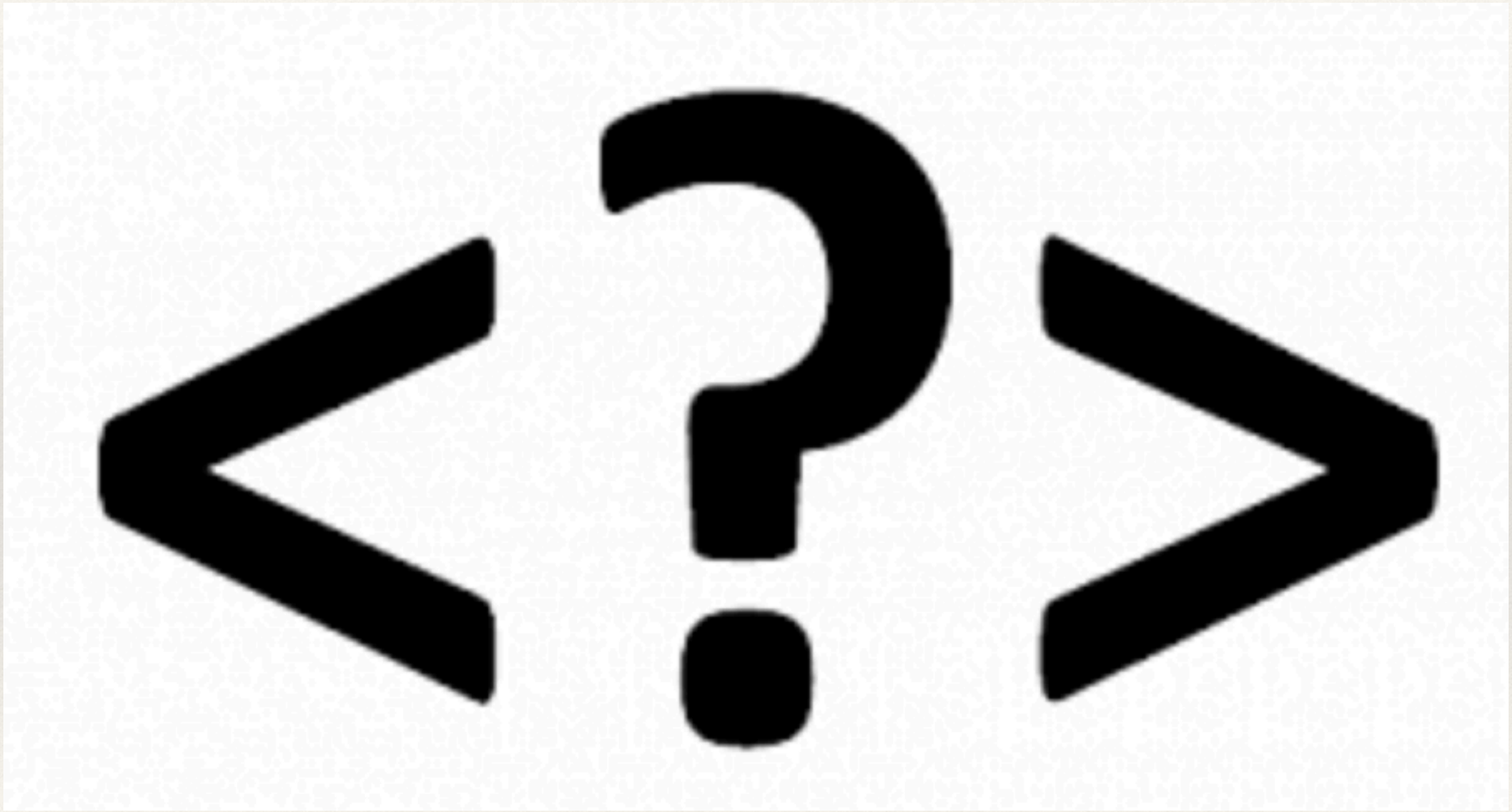
<R> Stream<R> map(Function<? super T,? extends R> transform)

Duck<? super Color> or Duck<? extends Color>

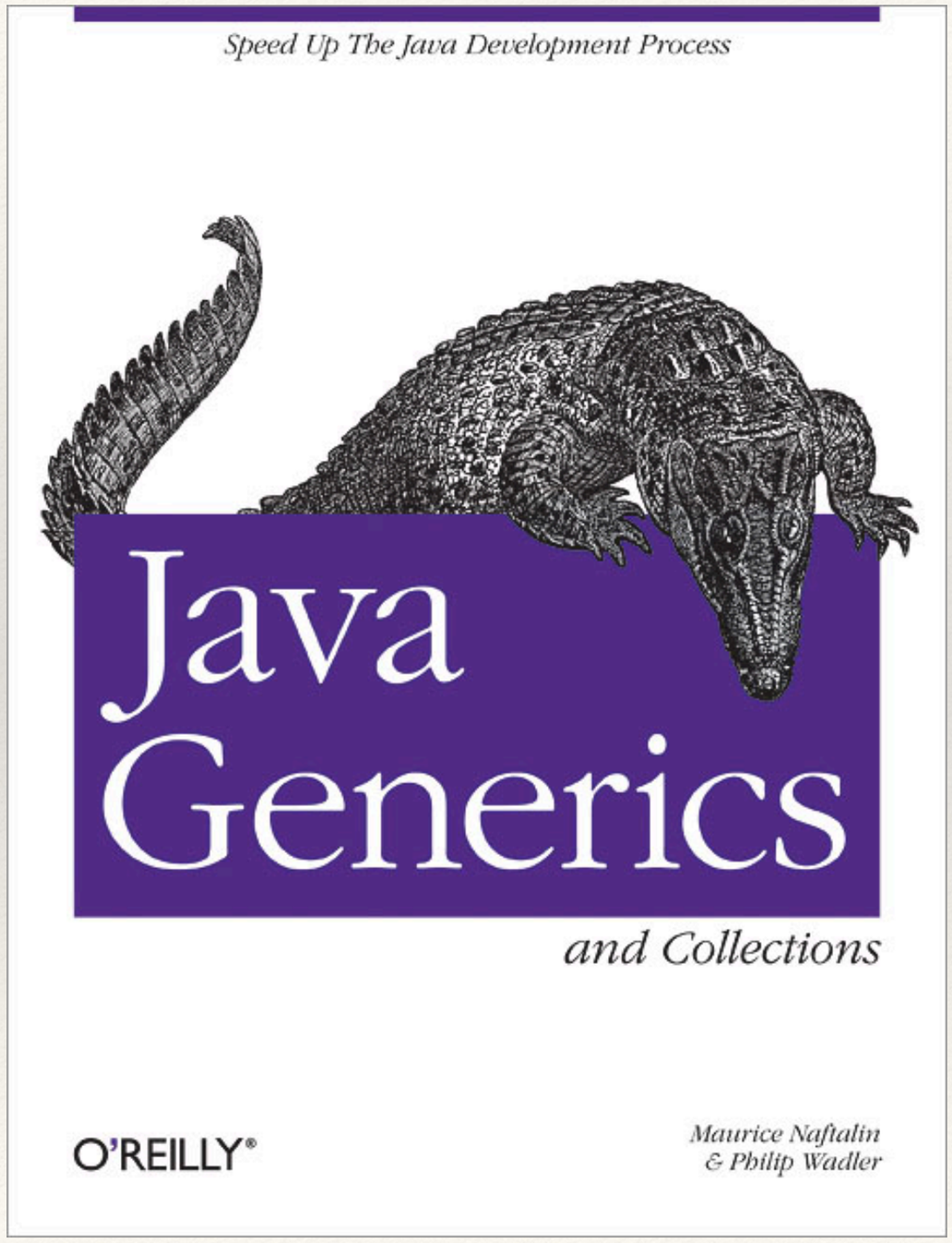
Generics: Code can be terrible to read



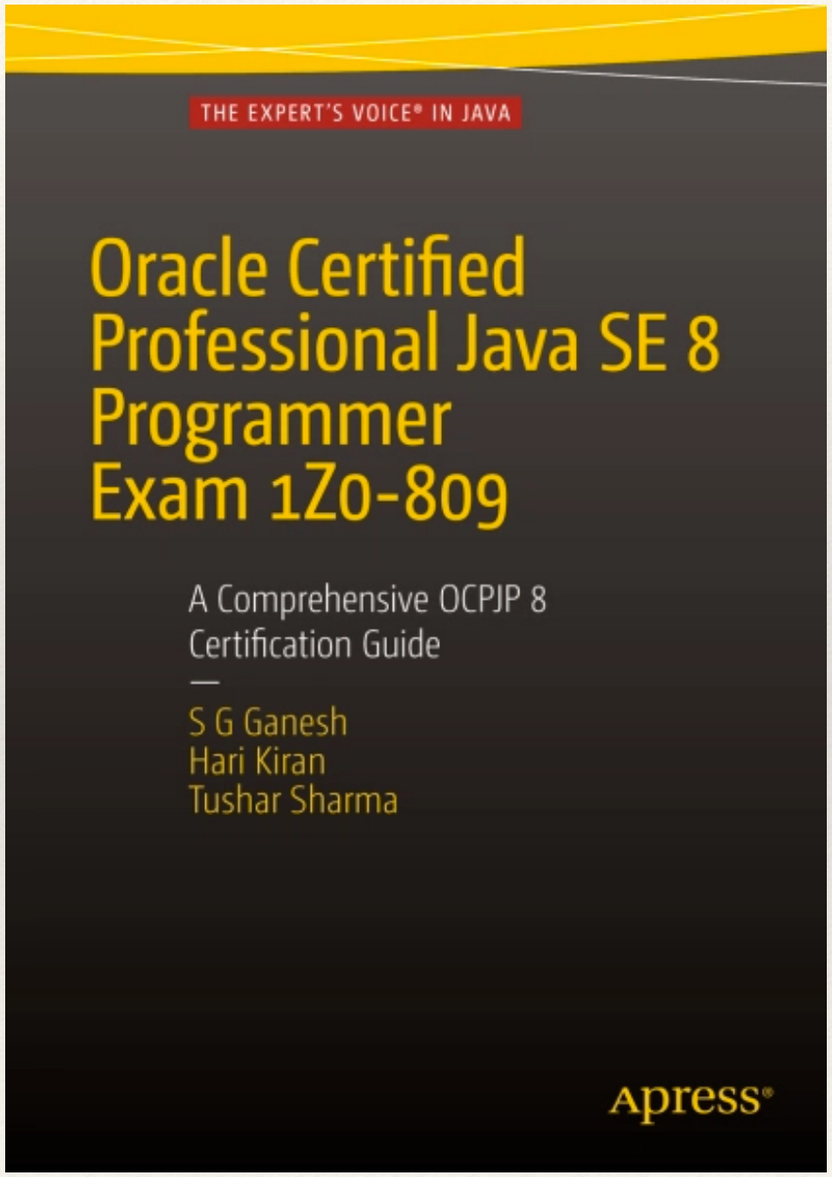
Question time



Books to read



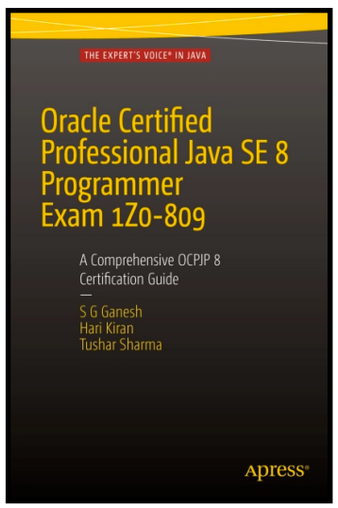
Books to read

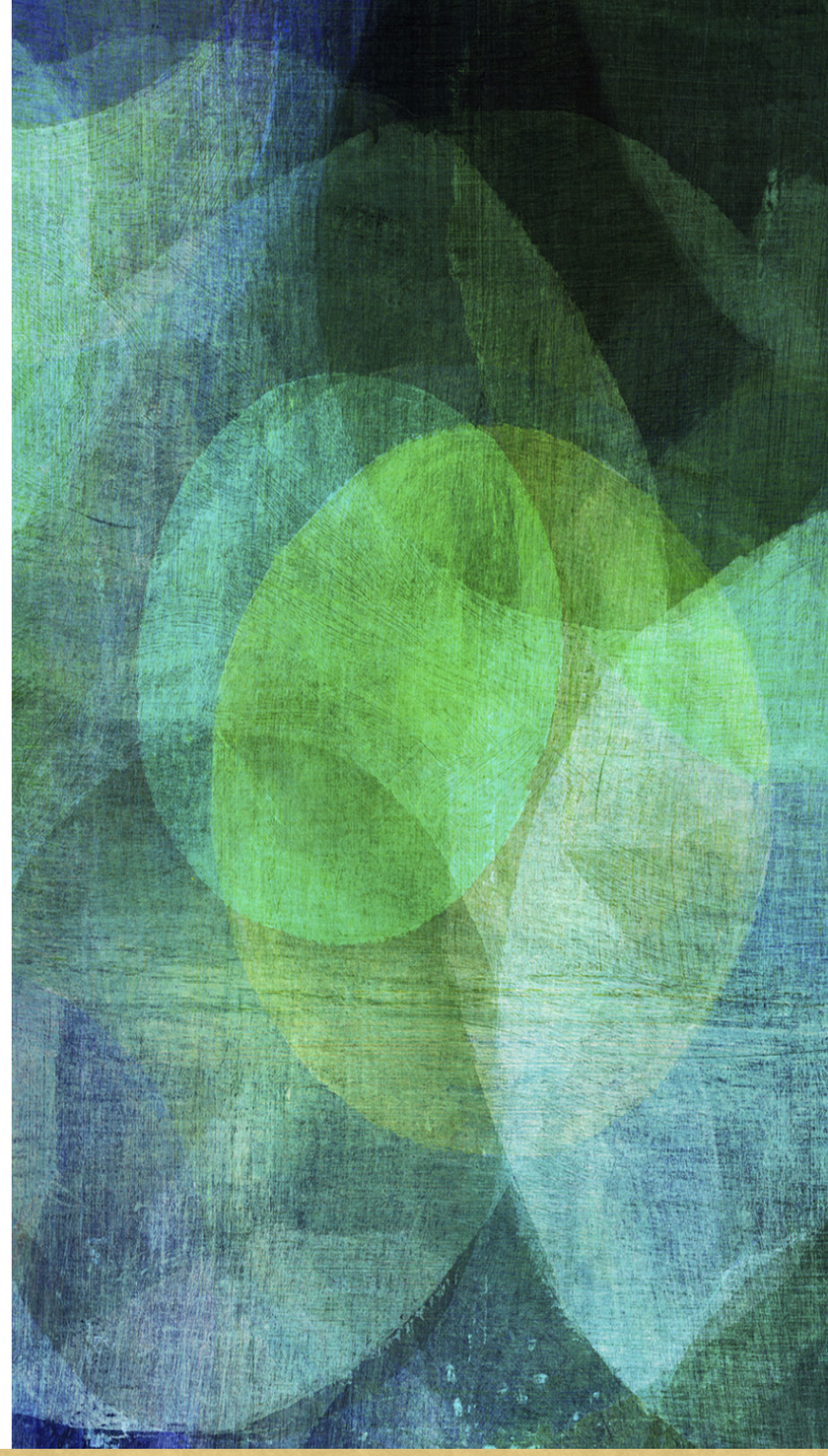


ocpjava.wordpress.com

Image credits

- http://www.ookii.org/misc/cup_of_t.jpg
- http://thumbnails-visually.netdna-ssl.com/tea-or-coffee_52612a16dfce8_w1500.png
- http://i1.wp.com/limpingchicken.com/wp-content/uploads/2013/04/question-chicken-1.jpg?resize=450%2C232
- https://4.bp.blogspot.com/-knt6r-h4tzE/VvU-7QrZCyI/AAAAAAAAAFYs/welz0JLVgbgOtfRSl98OxztA8Sb4VXxCg/s1600/Generics%2Bin%2BJava.png
- http://homepages.inf.ed.ac.uk/wadler/gj/gj-back-full.jpg
- http://kaczanowscy.pl/tomek/sites/default/files/generic_monster.jpeg
- https://s-media-cache-ak0.pinimg.com/736x/f7/b1/47/f7b147bfbc8332a2e3f2ba3c44a8e6d2.jpg
- https://avatars.githubusercontent.com/u/2187012?v=3
- http://image-7.verycd.com/fb2e74a15a36c4e7c1b6d4b1eb148af1104235/lrg.jpg
- http://regmedia.co.uk/2007/05/03/cartoon_duck.jpg
- https://m.popkey.co/48a22d/VWZR6.gif
- http://www.rawstory.com/wp-content/uploads/2012/07/boozehound-shutterstock.jpg
- http://www.fantasyfootballgeek.co.uk/wp-content/uploads/2016/04/wildcard-1-1.jpg
- https://i.ytimg.com/vi/XB0pGnzsAZI/hqdefault.jpg





ganesh@codeops.tech

www.codeops.tech

+91 98801 64463

@GSamarthyam

slideshare.net/sgganesh

bit.ly/sgganesh

CodeOps