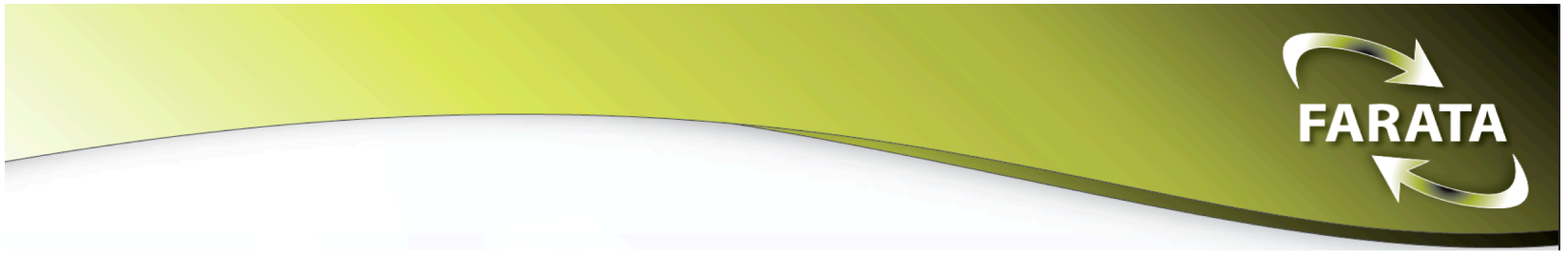


Java Programming
Unit 8

Selected Java Collections.
Generics.

Java Collections Framework

- Classes and interfaces from packages `java.util` and `java.util.concurrent` are called Java Collections Framework.

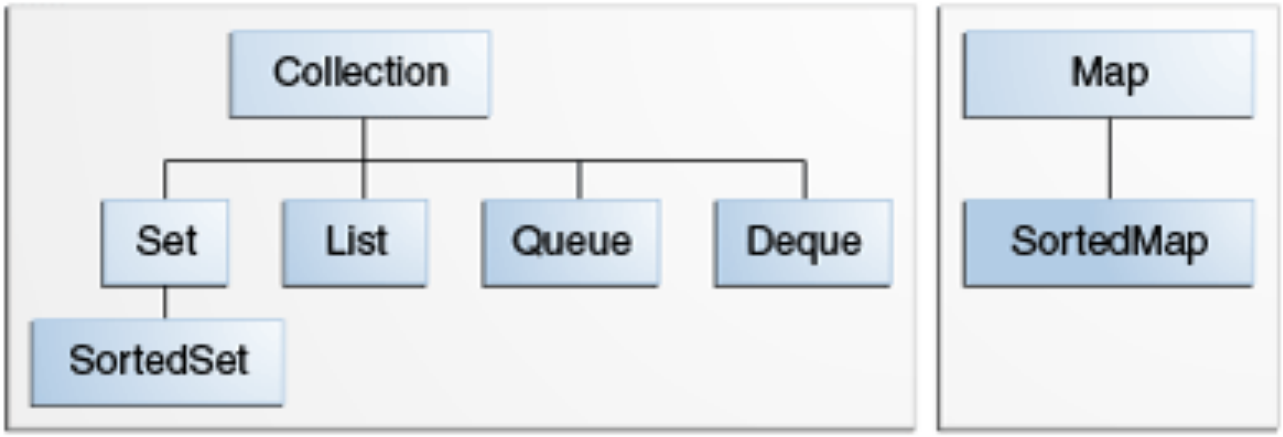
- java.util: http://bit.ly/1lXD3Kf

- java.util.concurrent: http://bit.ly/1iBREX5

- Collections store objects – no primitives allowed.

Java 8 improves collection iteration with `forEach()` and aggregate operations with `stream()`.
Details here: http://bit.ly/1iSzY9E

Core Collection Interfaces



This image is taken from Oracle documentation: http://bit.ly/1kV9EAh

Set, List, Queue, Map

- Sets cannot have duplicate elements. Implementations: `HashSet, TreeSet, LinkedHashSet`.
- Lists are ordered collections (sequences); lists can have duplicates and support positional access, iterations and search: `ArrayList, LinkedList.`
- Queues are first-in-first-out (FIFO) collections: `ArrayBlockingQueue, LinkedList.`
- Map objects mapkeys to values: `HashMap, TreeMap, LinkedHashMap`.

Populating an ArrayList

ArrayList is an unsynchronized resizable-array implementation of the List interface.

```
ArrayList customers = new ArrayList();

Customer cust1 = new Customer("David","Lee");
customers.add(cust1);

Customer cust2 = new Customer("Ringo","Starr");
customers.add(cust2);
```

`add()` doesn't copy instances of `Customer` obj into the collection `customers`, it just adds the memory addresses of the `Customer` instances.

You can specify the initial size of `ArrayList` by using one-argument constructor:

```
ArrayList customers = new ArrayList(10);
```

Getting objects from an `ArrayList`

The method get() extracts a particular `Object` from the `ArrayList`. You can cast it to the appropriate type.

```
ArrayList customers = new ArrayList();

customers.add(new Customer("David","Lee"));

Order ord = new Order(123, 500, "IBM");
customers.add(ord); // ????
```

Attempts to cast of Order to Customer, hence `IllegalClassCastException`

```
int totalElem = customers.size();

for (int i=0; i< totalElem;i++){
  Customer currentCust =
                (Customer) customers.get(i);
  currentCust.doSomething();
}
```

With **generics** you can do a compile-time check:

```
ArrayList<Customer> customers = new ArrayList<>(10);
```

Java 8 recommends iterating over collection with the new method `forEach()`.

Walkthrough 1 (start)

1. Download and import the source code for Lesson 14 into Eclipse

2. Add the following code to the end of the method main() in the class Test:

```
Order ord = new Order();
customers.add(ord);

int totalElem = customers.size();
for (int i=0; i< totalElem;i++){
    Customer currentCust =(Customer) customers.get(i);
}
```

3. Run Test and observe the runtime exception

4. Put the breakpoint (right-click | Toggle breakpoint) on the line

```
for (int i=0; i< totalElem;i++){
```

5. Debug the program. Observe the content of the variable `customers`.

continued…

Walkthrough 1 (end)

6. Modify the class Customer to look like this:

```
public class Customer {
    String firstName;
    String lastName;

    public Customer (String a, String b){
        firstName=a;
        lastName=b;
    }
}
```

7. Add the following line to the end of method main() in class Test:

```
System.out.println("The current customer is " + currentCust.lastName);
```

Why the program doesn't compile?

8. Move the println() line inside the for-loop and run the program.

9. Observe the output on the console and explain it:

```
The current customer is Lee
The current customer is Starr
Exception in thread "main" java.lang.ClassCastException: Order cannot be cast to Customer
        at Test.main(Test.java:23)
```

Hashtable and Hashmap are for key-value pairs

```
Customer cust = new Customer("David", "Lee");
Order ord = new Order(123, 500, "IBM");
Portfolio port = new Portfolio(123);

Hashtable data = new Hashtable();

data.put("Customer", cust);
data.put("Order", ord);
data.put("Portfolio", port);
```

Getting the object by key: `Order myOrder = (Order) data.get("Order");`

Hashtable is synchronized, but **Hashmap** is not (synchronization is explained in lesson 21). Consider synchronizing `HashMap` using `Collections.synchronizedMap(hashMap)`.

Hashtable is slow. Use `ConcurrentHashMap`.

Iterator Interface

```
Iterator iCust = customers.iterator();

   while (iCust.hasNext()){

       System.out.println( iCust.next() );
   }
```

`Iterator` can iterate the collection as well as remove items from it.

Starting form Java 8, use the method `forEach()` to iterate collections.

LinkedList

LinkedList is useful when you often need to insert/remove collection elements. Each element (a.k.a. node) contains a reference to the next one.

Insertion of a new object inside the list is a simple update of two references .

```
public class TestLinkedList {

 public static void main(String[] args) {

  LinkedList passengerList = new LinkedList();

  passengerList.add("Alex Smith");
  passengerList.add("Mary Lou");
  passengerList.add("Sim Monk");

  // Get the list iterator and print every element of the list
  ListIterator iterator =
              passengerList.listIterator();

  System.out.println(iterator.next());
  System.out.println(iterator.next());
  System.out.println(iterator.next());
 }
}
```

Java Generics

Generic type is the one that can have parameters.

For example, `ArrayList` is a generic type, but it allows you to specify a concrete parameter when it's instantiated.

Reading `ArrayList` Declaration

Open the doc for `ArrayList` at http://bit.ly/OunT0T :

```
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable,
                   java.io.Serializable
```

The **<E>** after the above class name tells the compiler that the type of *elements* to be stored in this class may be provided later, when the *concrete* instance of `ArrayList` is created, for example:

```
ArrayList<Customer> customers =
                    new ArrayList<>();
```

Diamond operator

Compile-Time Parameter Check

`ArrayList` can store any objects.

Do you want to store Cats and Dogs in the same `ArrayList`?

```
ArrayList<Customer> customers = new ArrayList<>();

Customer cust1 = new Customer("David","Lee");
customers.add(cust1);

Customer cust2 = new Customer("Ringo","Starr");
customers.add(cust2);

Order ord1= new Order();
customers.add(ord1); // Compiler error because of <Customer>
```

Getting a compiler's error is better than run-time class cast exceptions.

Iterating Parameterized ArrayList

```
ArrayList<Customer> customers = new ArrayList<>();

Customer cust1 = new Customer("David","Lee");
customers.add(cust1);

Customer cust2 = new Customer("Ringo","Starr");
customers.add(cust2);

// Iterate through the list customers and do something with each element

for (Customer c: customers){
    c.doSomething();   // no need to cast c from Object to Customer
                       // because of <Customer> parameter.
}
```

Walkthrough 2 (start)

1. Download and import the source code for the Lesson 15.

2. Run the program TestGenericCollection – it'll print the following:

   Customer David Lee. In doSomething()
   Customer Ringo Starr. In doSomething()

3. Un-comment the lines 16 and 17 to add an `Order` instance into the collection `customers`.

4. Observe the compiler error - can't add `Order` to the collection of `Customer` objects.

Walkthrough 2 (end)

5. Remove both `<Customer>` parameters from line 10. Compiler will stop complaining.

6. Run the program to see the **run-time** class cast exception. You've added the wrong object to the collection, but compiler didn't catch this error.

```
Exception in thread "main" java.lang.ClassCastException:
                                        Order cannot be cast to Customer
Customer David Lee. In doSomething()
Customer Ringo Starr. In doSomething()
        at TestGenericCollection.processData(TestGenericCollection.java:28)
        at TestGenericCollection.main(TestGenericCollection.java:23)
```

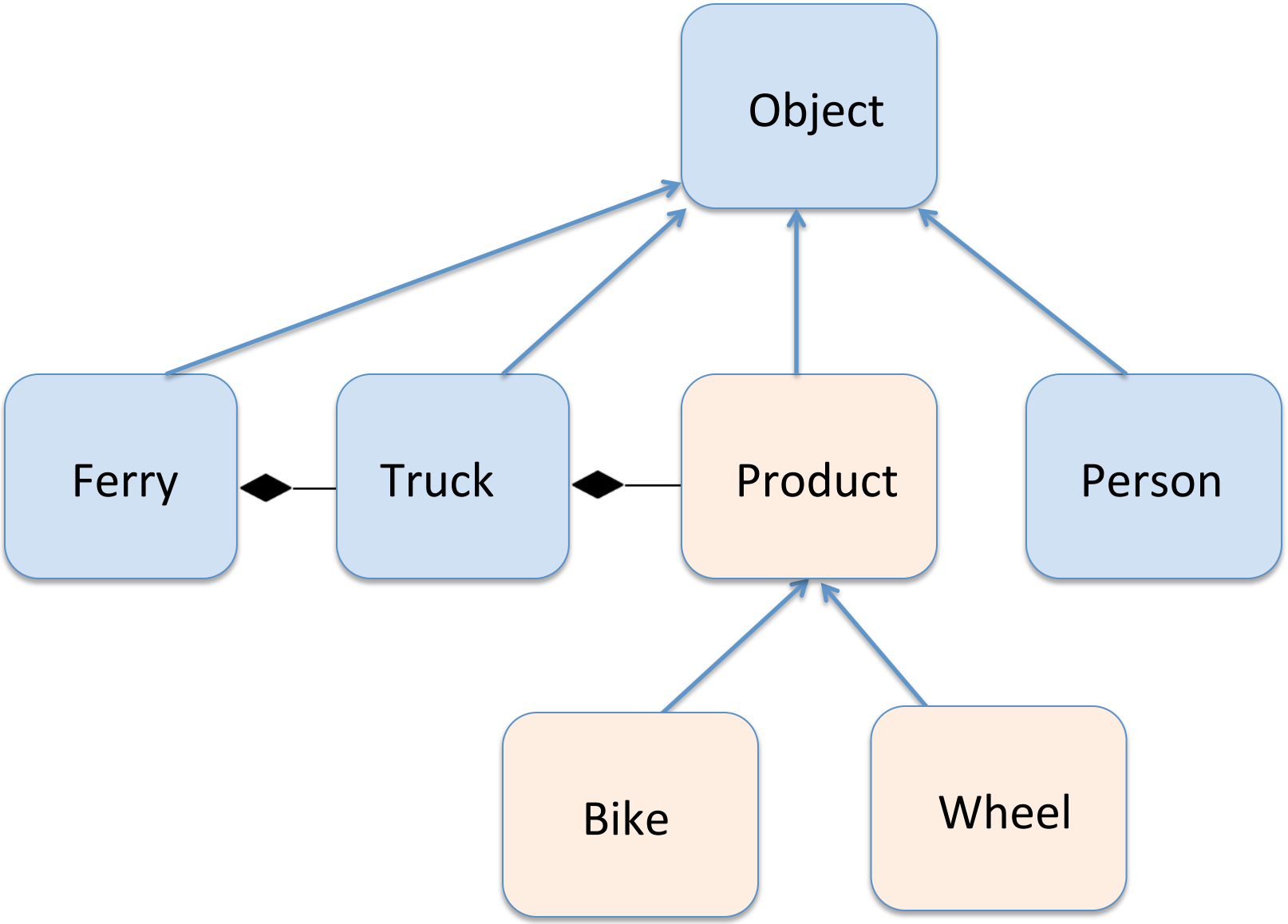
Defining Parameterized Classes

Below are the code snippets from the Oracle's Java Tutorial: http://bit.ly/1gDsOUj

```
public class Box<T> {

  // T stands for "Type"
   private T t;

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

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

```
public class BoxDemo3 {

 public static void main(String[] args) {

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

   integerBox.add(new Integer(10));
   Integer someInteger=integerBox.get(); // no cast!

    System.out.println(someInteger);

 }
}
```

We define a generic box to store objects of any type.
The concrete time will be provided by the user of the box.



Commonly Used Parameter Names

E - Element

K - Key

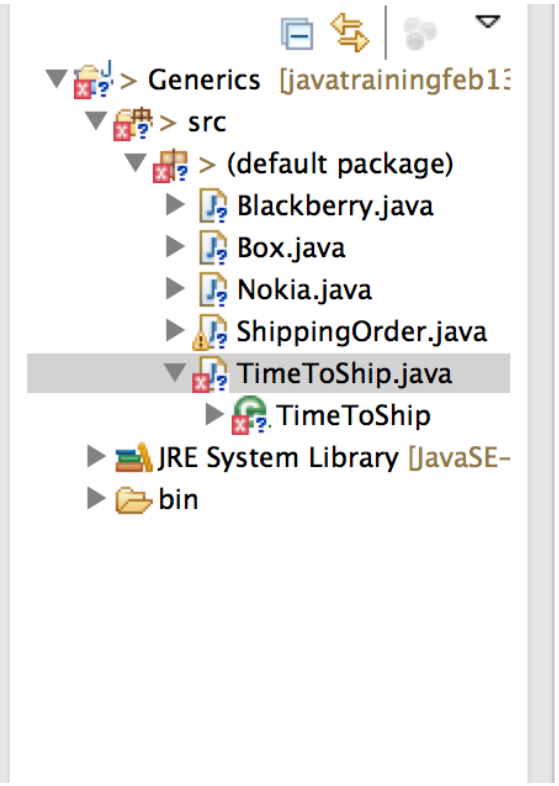
N - Number

T - Type

V - Value

Walkthrough 3

Let's find and fix the error in this code:




```
1  public class TimeToShip {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5
6          Box<Nokia> boxOfPhones = new Box<>();
7
8          // Create a shipment of Nokia phones
9          ShippingOrder<String, Box<Nokia>> so1=
10                 new ShippingOrder<> ("ph1234", boxOfPhones);
11
12         // Create a shipment of Blackberry phones
13         ShippingOrder<String, Box<Blackberry>> so2=
14                 new ShippingOrder<> ("ph4321", boxOfPhones);
15     }
16 }
17
```

Sources at https://github.com/yfain/javacodesamples

Type Erasure

- After insuring that programmer placed the proper types into a parameterized class, compiler erases all the info about parameters.

- For example, compiler will generate the same byte code (*raw type*) for these two types:

```
ArrayList<Customer> customers = new ArrayList<>();

ArrayList customers = new ArrayList();
```

- But the compiler will add required casting wherever customers is used.

Wildcards in Parameters

- `<?>` - unknown type
- `<? extends Customer>` - any type that extends Customer
- `<? super Customer>` - any type that's super class of `Customer`

```
private static void processData(
                          ArrayList<? extends Customer> customers)
{
       for (Customer c: customers){
          c.doSomething();
    }
}
```

Homework

Do the assignments from the Try It sections of Lesson 14 and 15

Additional Read

Linked lists: http://bit.ly/1gxCz5I

Study the Oracle's Java Generics Tutorial at http://bit.ly/1if4njs

Watch this presentation from the JavaOne conference on generics: http://bit.ly/14k7ORf

A simple example of using parameterized type <T> http://bit.ly/1mfsQsS