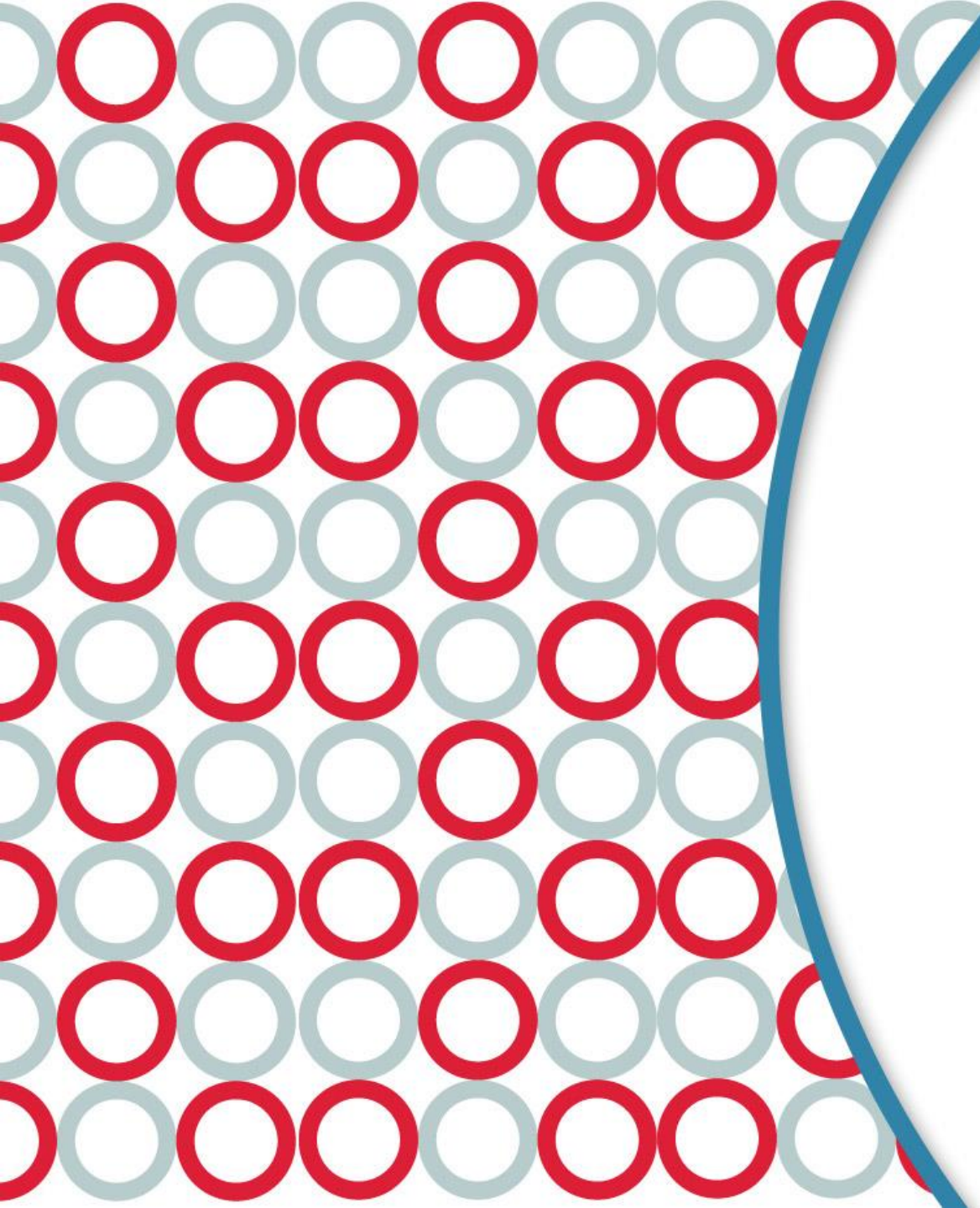


September 2020,
Programming in Java

Core Data Structures & Generics

Trayan Iliev

tiliev@iproduct.org
http://iproduct.org

About me

Trayan Iliev

– CEO of IPT – Intellectual Products & Technologies

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer

– Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Course Schedule

- Block 1: 13:40 – 15:10
- Pause: 15:10 – 15:20
- Block 2: 15:20 – 16:50
- Lunch: 16:50 – 17:00
- Block 3: 17:00 – 19:00 (18:00 on 14.09 and 17.09)

- Dates: 09 (Wed), 10 (Thu), 14 (Mon), 15 (Tue), 17 (Thu), 18 (Fri) September, 2020
- Project demonstrations: 23.09, 30.09, 18:00 - 19:30h

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

https://github.com/iproduct/course-java-fd

Agenda for This Session

- Arrays – immutability, manipulating elements, using utility methods of the class Arrays – sort, reverse, fill, copy, max, min, binarySearch, etc.
- List/ArrayList methods – get(), add(), remove(), set(), isEmpty(), size(), clear(), contains(), equals()
- Set/HashSet methods
- Map/HashMap methods
- Generic Types

Arrays. Comapring and Sorting

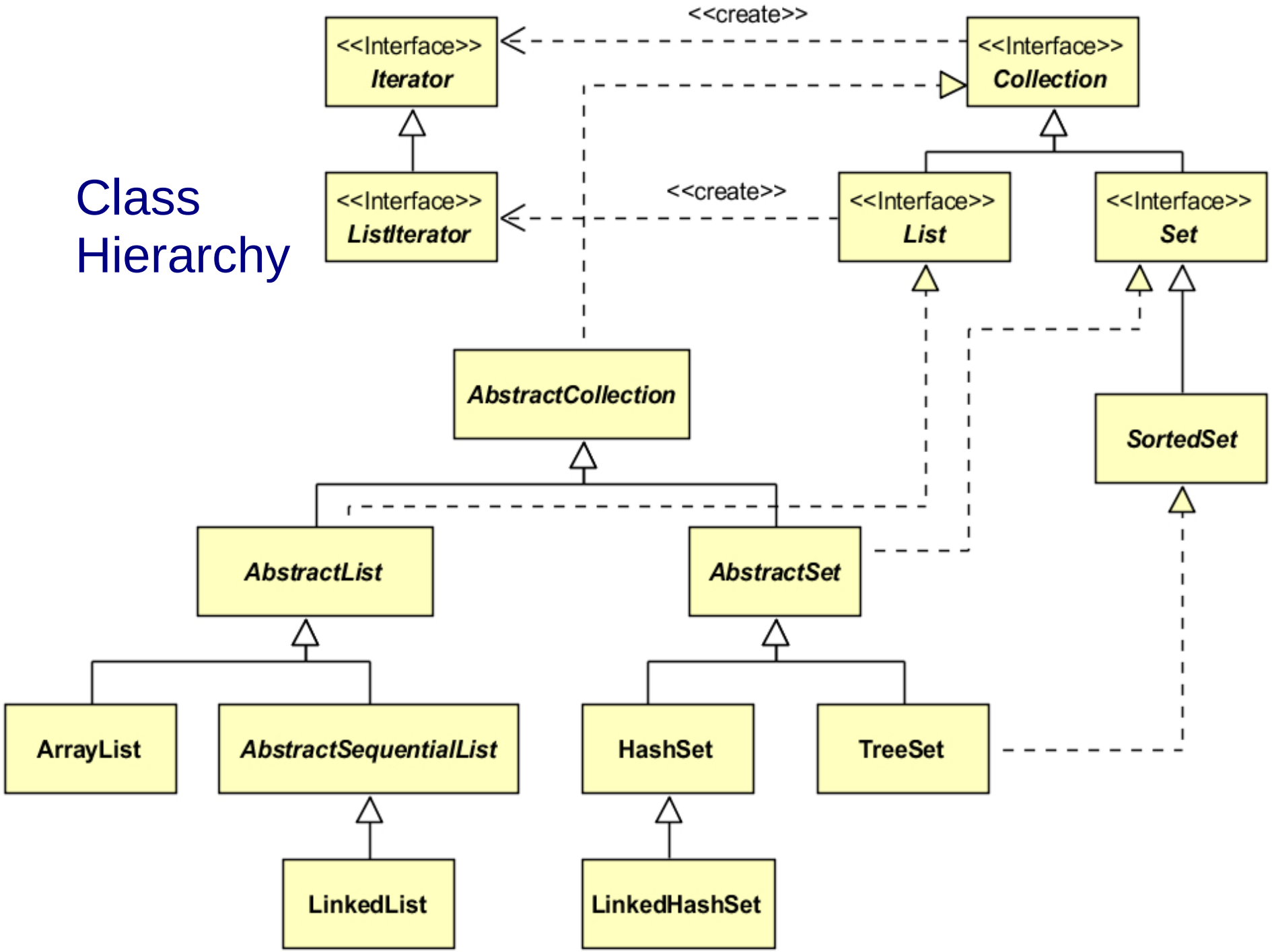
- Arrays and working with them
- Utility methods of the class **Arrays**:

  – equals()
  – fill()
  – copyOf() и copyOfRange()
  – binarySearch()
  – sort()
- Comparing objects – interfaces **Comparable** and **Comparator**

Container Classes and Interfaces. Iterators.

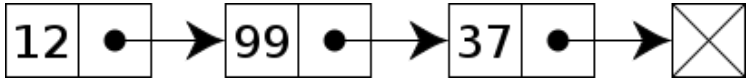
- Колекции – интерфейс **Collection**
- Списъци – интерфейс **List**, реализации – **ArrayList**, **LinkedList**, ...
- Множества – интерфейс **Set**, реализации – **HashSet**, **TreeSet**, …
- Асоциативни списъци – интерфейс **Map**, реализации – **HashMap**, **TreeMap**, **LinkedHashMap**, **WeakHashMap**, …
- Обхождане на колекция с итератор.
- Реализиране на структури от данни стек, опашка, дек – интерфейси **Queue** и **Dequeue**. Реализации.

Class Hierarchy

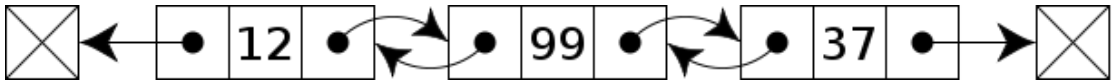


Data Structures

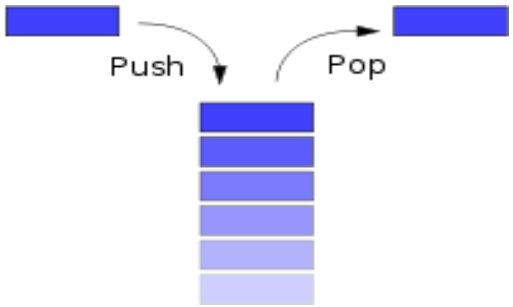
- Linked list:



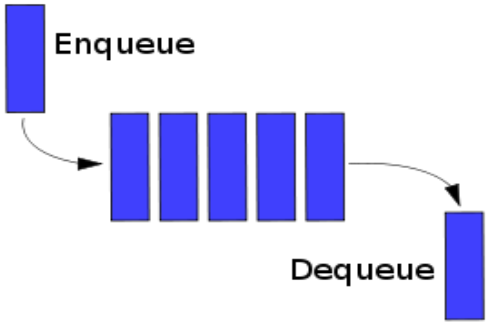
- Doubly-linked list:



- Stack:

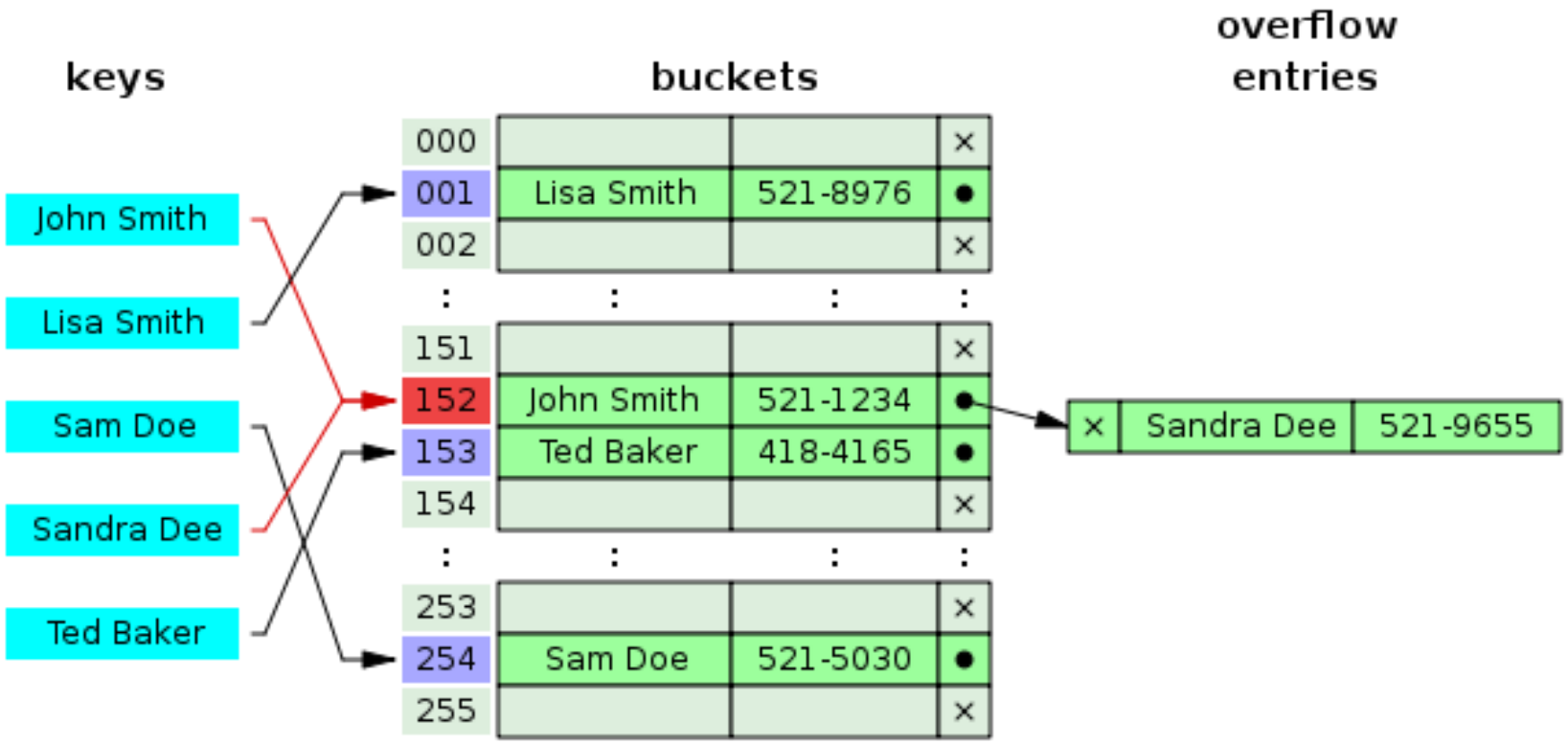


- Queue:



Source: Wikipedia, http://en.wikipedia.org/

Hashinng. Hash-Functions. Hash Tables



Source: http://en.wikipedia.org/wiki/File:Hash_table_5_0_1_1_1_1_0_LL.svg

Parameterizied Types: Generics (1)

- Collections and their methods before Java 5 were limited to handle a single type of elements.
- If we want to create typed containers we had to implement different container types for each entity type.
- *Example:* In a e-Bookstore we want to sell **Books** and want the container to contain only **Books** (being strongly typed) --> we should implement separate class **BookList**, as well as for each **Book** we want to keep a list of **Authors** --> we should implement **AuthorList** too, and so on.

Parameterizied Types: Generics (2)

- *Solution:* We can skip writing multiple similar classes (e.g. typed containers for each type of elements) using **Generic types**

- *Generic type invocation:*

List<Book> books = new ArrayList<Book>()

List<Author> authors = new ArrayList<Author>()

- **<> – *Diamond*** *operator* – new in Java™ 7, allows automatic inference of the generic type:

List<Book> books = new ArrayList<>()

List<Author> authors = new ArrayList<>()

Parameterizied Types: Generics (3)

- *Generic type declaration:*

```
public class Position<T extends Product> {
        private T product;
        ...
        public Position(T product, double quantity) {
                this.product = product;
                this.quantity = quantity;
                price = product.getPrice();
        }
        public T getProduct() {
                return product;
        }
...
```

Generic data type

Conventions Naming Generic Parameters

- Generic parameters naming conventions:

T – type parameter (if there are more – S, U, V, W ...)

E – element of a collection – e.g.: List<E>

K – key in associative pair – e.g.: Map<K,V>

V – value in associative pair – e.g.: Map<K,V>

N – number value

Example:

```
public class Invoice <T extends Product> {
        ...
        private List<Position<T>> positions = new ArrayList<>();
    ...
}
```

Generic Methods (1)

- We can implement generic methods and constructors too:

```
public static <U extends Product> String
getPositionsAsString (List<Position<U>> positions) {
    StringBuilder posStr = new StringBuilder();
    int n = 0;
    for(Position<U> p: positions){
        posStr.append( String.format(
"\n| %1$3s | %2$30s | %3$6s | %4$4s | %5$6s |%6$8s |",
++n, p.getProduct().getName(), p.getQuantity(),
p.getProduct().getMeasure(),p.getPrice(), p.getTotal()
        ));
    }
    return posStr.toString();
} ...
```

Generic Methods (2)

- Invoking generic method / constructor:

```
result += Invoice.<T> getPositionsAsString(positions);
```

- OR we can let Java to automatically infer the generic type:

```
result += Invoice.getPositionsAsString(positions);
```

Bounded Type Parameters

- We can define upper bound constraint for the possible types that can be allowed as actual generic type parameters of the class / method /constructor:

```
public static <U extends Product> String
getPositionsAsString (List<Position<U>> positions) { ... }
```

- OR

```
public static <U extends Product & Printable> String
getPositionsAsString (List<Position<U>> positions) {
    ...
    p.getProduct().print();
    ...
}
```

Generics Sub-typing

- If the class Product extends class Item, can we say that List<Product> extends List<Item> too? Can we substitute the first with the second?
- The answer is „**NOT**“, because the basic generic type is not designed to reflect the specifics of of the Products.
- Dos and donts when using generics inheritance:

```
interface Service extends Item; Service s = new Service( ...);
Collection<Service> services =  ...; services.add(s); // OK
interface Product extends Item; Product p = new Product( ...);
Collection<Product> products =  ...; products.add(p); // OK
Collection<Item> items = ...; items.add(s); items.add(p); // OK
items = products; // NOT OK
items = services; // NOT OK
```

Using ? as Type Specifier (Wildcards)

- If we want to declare that we expect specific, but not pre-determined type, which for example extends the class **Item**, we could use **?**:

```
Collection<? extends Item> items; // Upper bound is Item
items = products; // OK
items = services; // OK
Items.add(p); // NOT OK – Can not write into it – it is not safe!
Items.add(s); // NOT OK – Can not write into it – it is not safe!
for(Item i: items) { // OK – Can read it – it is known to be at least Item.
    System.out.println( i.getName() + „:“ + i.getPrice() );
}
List<? super Product> products; // Lower bound is Product
products.add(p); // OK – Can write into it – it is now safe.
Product p = products.get(0); //NOT OK may be superclass of Product
```

- Producer extends and Consumer super (PECS) principle

Type Erasure & Reification

- **Type Erasure** – chosen in java as backward-compatibility alternative – information about generic type parameters is erased during compilation, and is NOT available in runtime – the generic type becomes compiled to its basic raw type:

  Collection<Product> products; --(runtime)--> Collection products;

  This design decision creates problems if we want to create generic type instance with **new**, or to convert to the generic type, or to check the generic type using **instanceof**.

- **Reification** – better alternative strategy, implemented in languages such as C++, Ada и Eiffel, using which the generic type information is accessible in runtime.

Generic Containers

- Allow compile time type checking – earlier error detection
- Remove unnecessary typecasting to more specific types – less `ClassCastExceptions`
- Examples:

```
Collection <String> s = new ArrayList <String>();
Map <Integer, String> table = new HashMap <Integer, String>()
```

- New for loop – for each element of a Collection :

```
for(String i: s) { System.out.println(i) }
```

Main Implementing Classes. Examples

- Associative lists (dictionaries) – interface **Map**
- Comparing different implementations:
  - **HashMap**
  - **TreeMap**
  - **LinkedHashMap**
  - **WeakHashMap**
- Hashing.
- Cash implementations – **Reference, SoftReference, WeakReference и PhantomReference**
- Choosing a container implementation

Литература и интернет ресурси

- Oracle Generics tutorial – https://docs.oracle.com/javase/tutorial/extra/generics/index.html

Thank's for Your Attention!

Trayan Iliev

CEO of IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD