Εργαστήριο 01α: Επανάληψη Αντικειμενοστρεφή Προγραμματισμού με JAVA

Πέτρος Παναγή

Άσκηση 1

Θέλουμε να δημιουργήσουμε την κλάση **`IntegerSet`**, η οποία αναπαριστά ένα σύνολο από αριθμούς. Κάθε αντικείμενο τύπου IntegerSet περιλαμβάνει αριθμούς από το πεδίο **0 - 100**. Το σύνολο των αριθμών αυτών αναπαριστάται με την μορφή ενός **πίνακα τύπου booleans.** Το στοιχείο του πίνακα a[i] είναι αληθές εάν ο ακέραιος i ανήκει στο σύνολο. Το στοιχείο του πίνακα a[i] είναι ψευδές εάν ο ακέραιος i δεν ανήκει στο σύνολο. Ακολουθούν οι λεπτομέρειες για την υλοποίηση της κλάσης:

- Η κλάση θα έχει δυο χαρακτηριστικά τα οποία ονομάζουμε, **`SETSIZE (int)`** και **`set_elements`** (πίνακας από boolean).
- Γράψτε ένα **κατασκευαστή** χωρίς παραμέτρους ο οποίος θα αρχικοποιεί το σύνολο ως το κενό σύνολο.
- Προσθέστε μια μέθοδο **`insertElement`** η οποία παίρνει σαν παράμετρο ένα ακέραιο k και τον εισαγάγει στο σύνολο.
- Προσθέστε μια μέθοδο **`deleteElement`** η οποία παίρνει σαν παράμετρο ένα ακέραιο k και τον διαγράφει από στο σύνολο.

Άσκηση 1

- Προσθέστε μια μέθοδο **`toSetString`** η οποία εκτυπώνει στην οθόνη μόνο τους αριθμούς που ανήκουν στο σύνολο διαχωριζόμενους από τον χαρακτήρα ' '. Χρησιμοποιείστε τους χαρακτήρες { } για να αναπαραστήσετε το κενό σύνολο.
- Γράψτε μια μέθοδο **public boolean `isEqualTo(IntegerSet` is2)** η οποία καθορίζει εάν τα δύο σύνολα είναι ίσα.
- Γράψτε μια μέθοδο public **`IntegerSet union(IntegerSet is2)`** η οποία δημιουργεί ένα τρίτο σύνολο το οποίο αποτελεί την ***ένωση*** των δύο συνόλων.
- Γράψτε μια μέθοδο public **`IntegerSet intersection(IntegerSet is2)`** η οποία δημιουργεί ένα τρίτο σύνολο το οποίο αποτελεί την ***τομή*** των δύο συνόλων.

Δημιουργήστε το αρχείο **IntegerSet.java** το οποίο περιέχει όλα όσα περιγράφονται πιο πάνω. Συμπληρώστε την συνάρτηση main για να ελέγξετε την κλάση IntegerSet. Για το έλεγχο αυτό δημιουργήστε διάφορα αντικείμενα τύπου IntegerSet και βεβαιωθείτε ότι όλες οι μέθοδοι λειτουργούν ορθά.

package cy.ac.ucy.cs.epl231.ID123456.Lab1

Εργαστήριο 01β:
Προγραμματισμός με Γενικούς Τύπους
(JAVA Generics)

https://docs.oracle.com/javase/tutorial/java/generics/

Πέτρος Παναγή

E. Dijkstra - 1972

In 1972 **E. Dijkstra** during a Turing Award Lecture has stated that:

“as long as there were no machines, programming was no problem at all;

when we had a few weak computers, programming became a mild problem,

and now we have gigantic computers, programming has become an equally gigantic problem."

Προγραμματισμός Υπολογιστών

"First Software Crisis" in the 60's and 70's

Computer programming was done using **Machine Code** and **Assembly**.

Fortran and **C** made their appearance to resolve this crisis.

"Second Software Crisis" in the 80's and 90's

The inability to build and maintain complex and robust applications like Operading Systems, DB etc..

Object Oriented Programming came as the solution (C++, Java, C#)

"Third Software Crisis" in the 00's until today

Parallel Computes-Multi Processor – Multi Core Machine

Autoboxing and Unboxing

- **Autoboxing** είναι η αυτόματη μετατροπή που κάνει ο compiler της Java μεταξύ των πρωτόγονων τύπων (Primitive Types) και κατηγοριών τους περιτύλιγμα αντίστοιχο αντικείμενο (object wrapper classes) . Για παράδειγμα , μετατροπή ενός `int` σε `Integer`, ένα `double` σε ένα `Double`, και ούτω καθεξής. Αν η μετατροπή πάει τον άλλο τρόπο , αυτό ονομάζεται **unboxing**.
- Autoboxing και unboxing επιτρέπει στους προγραμματιστές να γράφουν καθαρότερο κώδικα , πράγμα που καθιστά ευκολότερη την ανάγνωση του κώδικα. Ο παρακάτω πίνακας παραθέτει τους πρωταρχικούς τύπους και τις αντίστοιχες wrapper classes, τα οποία χρησιμοποιούνται από τον compiler της Java για autoboxing και unboxing :

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

java.lang.Iterable

Η διεπαφή Iterable (`java.lang.Iterable`) είναι μία από τις διασυνδέσεις ρίζα (root) των κατηγοριών συλλογής Java. Η διεπαφή `Collection` εκτείνεται (extends) `Iterable` , έτσι ώστε όλοι οι υπότυποι της `Collection` υλοποιήσει (Implement) το `Iterable`.

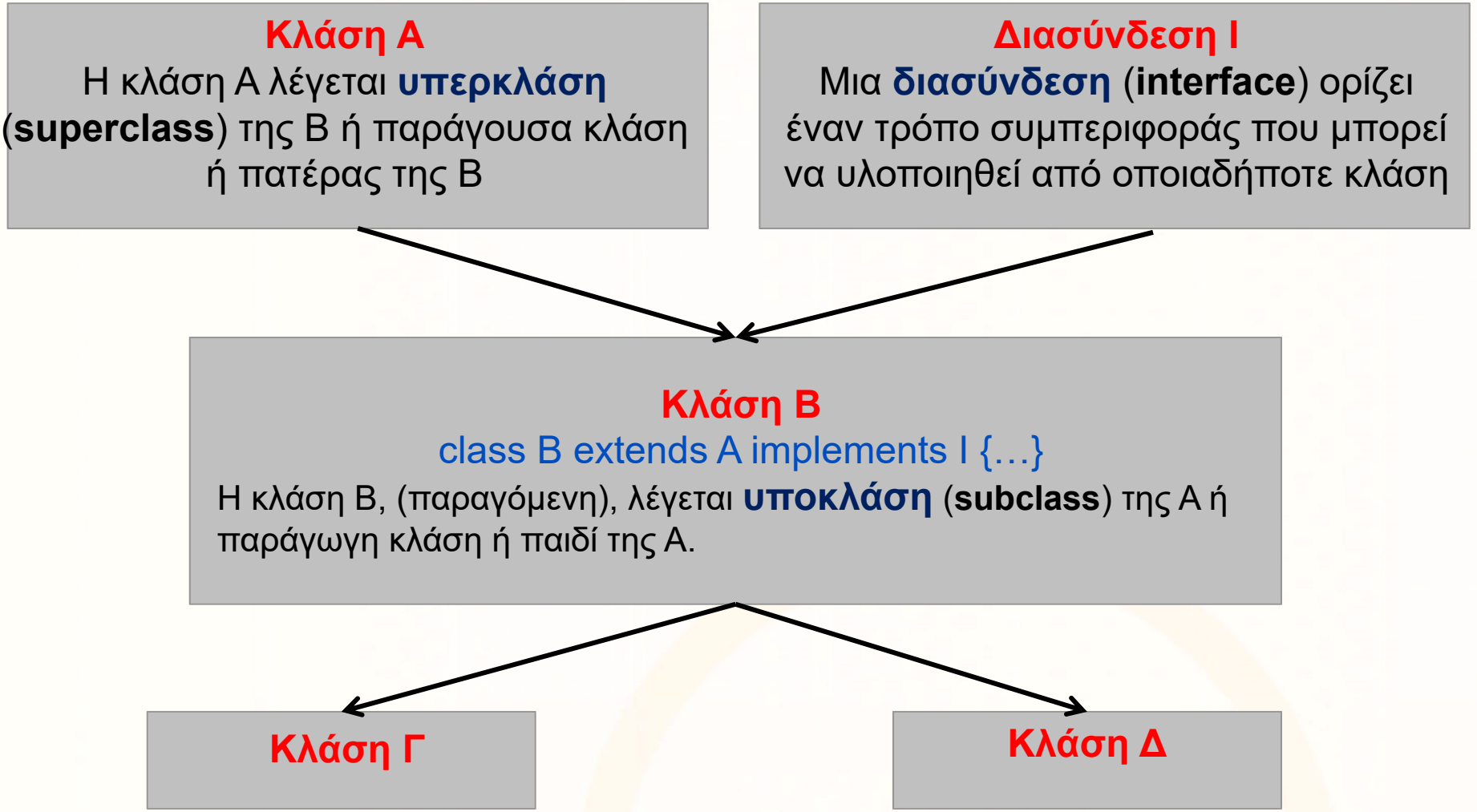
Μια κλάση που υλοποιεί το `Iterable` μπορούν να χρησιμοποιηθούν με το νέο for-loop. Για παράδειγμα :

```
list = new ArrayList();
for(Object o : list){
    //do something with o;
}
```

The Iterable interface has only one method:

```
public interface Iterable<T> {
  public Iterator<T> iterator();
}
```

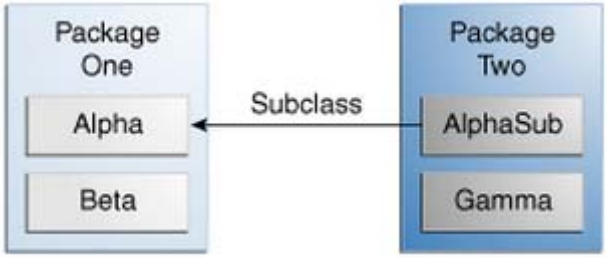
Κληρονομικότητα Ορολογία



Controlling Access to Members of a Class

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N



Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

static keyword to create fields and methods that belong to the class, rather than to an instance of the class.

Every instance of the class shares a class **variable**, which is in one fixed location in memory.

Static methods, should be invoked with the class name, without the need for creating an instance of the class

The **static** modifier, in combination with the **final** modifier, is also used to define constants.

```
static final double PI =
3.141592653589793;
```

Generic Types

- Τα Generics έχουν προστεθεί στη JAVA από το 2004 ως μέρος του J2SE 5.0
- Με τη χρήση των generics μπορούμε να δημιουργήσουμε κλάσεις και μεθόδους που να διαχειρίζονται διαφόρων ειδών αντικείμενα (π.χ., Στοίβα με ακέραιους, Στοίβα με συμβολοσειρές) με την ίδια υλοποίηση.

- Ο προγραμματισμός με generics μπορεί να γίνει αρκετά περίπλοκος
- Εμείς, θα επικεντρωθούμε στα ιδιώματα και την σύνταξη που παρουσιάζονται στο βιβλίο.
- Δεν θα μπούμε σε βάθος που να καλύπτει όλο το εύρος της χρήσης τους

Μία γενική κλάση MemoryCell (πριν την JAVA 5.0)

```
// MemoryCell class
// Object read( ) --> Returns the stored value
// void write( Object x ) --> x is stored
public class MemoryCell {
        // Private internal data representation
        private Object storedValue;

        // Public methods
        public Object read() {
                return storedValue;
        }

        public void write(Object x) {
                storedValue = x;
        }
}
```

Μία γενική κλάση MemoryCell (με Generics)

```
// MemoryCell class
// AnyType read( ) --> Returns the stored value
// void write( AnyType x ) --> x is stored
public class MemoryCell<AnyType> {
        // Private internal data representation
        private AnyType storedValue;

        // Public methods
        public AnyType read() {
                return storedValue;
        }

        public void write(AnyType x) {
                storedValue = x;
        }
}
```

Γιατί να χρησιμοποιηθούν τα generics;

Μία κλάση generic έχει την πιο κάτω μορφή:

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

Ο κώδικας που χρησιμοποιεί generics έχει πολλά πλεονεκτήματα:

- **Καλύτερος έλεγχος σε επίπεδο μεταγλώττισης**
  Ο μεταγλωττιστής της JAVA εφαρμόζει ισχυρό έλεγχο για τύπους. Έτσι τα λάθη διορθώνονται σε επίπεδο μεταγλώττισης και είναι πιο δύσκολο να δημιουργηθούν runtime exceptions

- **Απάλειψη των μετατροπών τύπων (casting)**
  - Παράδειγμα χωρίς generics: `List list = new ArrayList(); list.add("hello"); String s = (String) list.get(0);`
  - Παράδειγμα με generics: `List<String> list = new ArrayList<String>(); list.add("hello"); String s = list.get(0); // no cast`

- **Επιτρέπει την δημιουργία γενικών αλγορίθμων**
  Οι προγραμματιστές μπορούν να δημιουργούν γενικούς αλγόριθμους που δουλεύουν σε συλλογές διαφορετικών τύπων που είναι ασφαλής όσο αφορά τους τύπους (type safe) και είναι πιο κατανοητές

Κοινοί τύπου Generics

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- **T - Type**
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

- Αυτοί οι τύποι χρησιμοποιούνται σε διάφορες βιβλιοθήκες του Java API

Δημιουργία και Αρχικοποίηση ενός τύπου Generic

- Για δημιουργία και αναφορά ενός τύπου Generic στον κώδικα, θα πρέπει να αντικαταστήσετε τον τύπο Generic (π.χ., E, T) με ένα "γνωστό/κανονικό" τύπο (π.χ., Integer).
  Παράδειγμα:

```
MemoryCell<Integer> intMemoryCell;
```

- Προσοχή: μπορούν αν χρησιμοποιηθούν μόνο αναφορές σε αντικείμενα (δηλ., Integer και όχι ο αρχέγονος τύπος int)

- Παρόμοια με άλλες δηλώσεις μεταβλητών, ο πιο πάνω κώδικας έχει δημιουργήσει μία **αναφορά** σε ένα αντικείμενο τύπου MemoryCell object. Για να αρχικοποιηθεί το **αντικείμενο** θα πρέπει να χρησιμοποιηθεί το keyword **new**, αλλά επίσης να δηλωθεί ο τύπος που θα χρησιμοποιηθεί (δηλ., Integer) πριν τις παρενθέσεις ():

```
MemoryCell<Integer> intMemoryCell = new
MemoryCell<Integer>();
```

Δημιουργία και Αρχικοποίηση ενός τύπου Generic

- Από την ***Java SE 7***, κατά τη διαδικασία αρχικοποίησης, μπορούν να αντικατασταθούν οι παράμετροι τύπου με το άδειο σύνολο (<>) δεδομένου ότι ο μεταγλωττιστής μπορεί να ερμηνεύσει τον τύπο από την δήλωση
  Παράδειγμα:

```
MemoryCell<Integer> intMemoryCell = new MemoryCell<>();
```

- Οι παρενθέσεις **<>**, ονομάζονται και *the diamond*.

Generic Interfaces

- Η διαπροσωπεία **Comparable**, Java v.x < 5

```
package java.lang;
public interface Comparable {
      public int compareTo( Object o );
}
```

- Η διαπροσωπεία Comparable, Java v.x >= 5

```
package java.lang;
public interface Comparable<T> {
      public int compareTo( T o );
}
```

Οριοθέτηση των Παραμέτρων

Ενδέχεται να υπάρχουν φορές που θέλουμε να περιορίσουμε τα εύρος των τύπων που μπορούν να χρησιμοποιηθούν σε ένα αντικείμενο generic.

Για παράδειγμα, μια μέθοδος που λειτουργεί σε `Numbers` μπορεί να θέλει να δεχτεί μόνο στιγμιότυπα του `Numbers` ή των `subclasses` του.

Αυτές είναι οι `bounded type parameters`. Για το σκοπό αυτό χρησιμοποιούμε την λέξη `extends`.

```
public class NaturalNumber<T extends Integer> {

    private T n;

    public NaturalNumber(T n)  { this.n = n; }

    public boolean isEven() {
        return n.intValue() % 2 == 0;
    }

    // ...
}
```

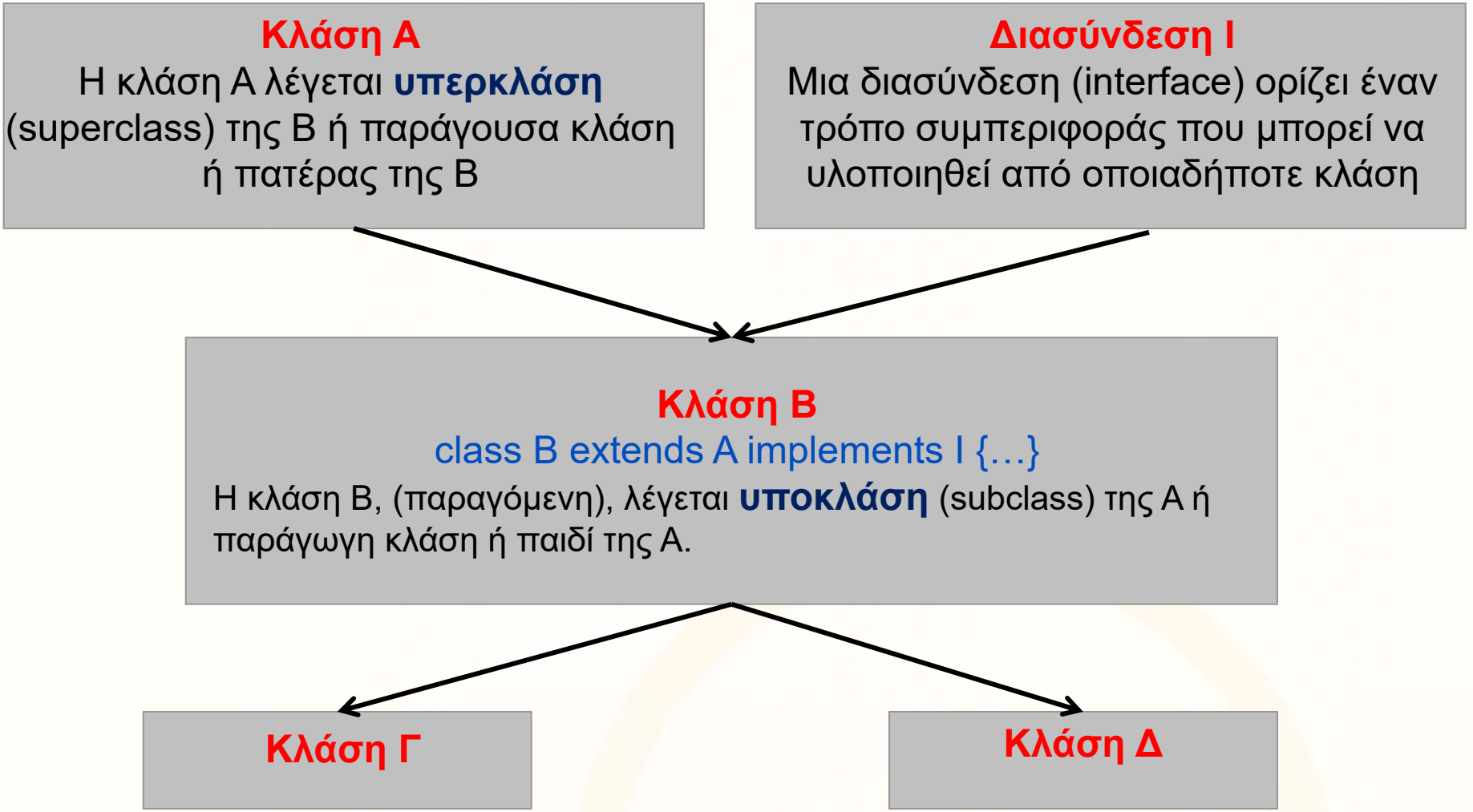
Τύποι Μπαλαντέρ (Wildcards <?>) (για Παραμέτρους)

- Το **wildcard** μπορεί να χρησιμοποιηθεί σε διάφορες καταστάσεις: ως **τύπος παραμέτρου**, **πεδίου**, **τοπικής μεταβλητής**.
- Υποθέστε μία κλάση **Shape** και μία μέθοδο `double area()`
- Υποθέστε επίσης ότι υπάρχει μία στατική μέθοδος **`totalArea()`** που δέχεται μία συλλογή (Collection) από Shapes (δηλ., **Collection<Shape>**) και βρίσκει το συνολικό εμβαδό όλων των σχημάτων της συλλογής

```
public static double totalArea (Collection<Shape> arr) {
      double total = 0.0;
      for ( Shape s : arr ) //For-Each Shape s in arr
              if ( s != null ) total += s.area();
      return total;
}
```

- Τώρα, υποθέστε δύο κλάσεις **Circle** και **Square** που κληρονομούν την κλάση Shape.
- **Ερώτηση:** Τι θα συμβεί αν περάσουμε σαν παράμετρο μία συλλογή τύπου **Collection<Square>**;

Κληρονομικότητα Ορολογία



Χρήση Wildcards <?>

- **Απάντηση:** Λάθος μεταγλώττισης.
- Οι Generic συλλογές δεν είναι γενικού τύπου (covariant).
- Συνεπώς, δεν μπορούμε να περάσουμε σαν παράμετρο αναφορά σε αντικείμενο τύπου Collection<Square> στη μέθοδο totalArea()
- Τα Wildcards <**?**> χρησιμοποιούνται για να αναφερθούμε σε υποκλάσεις (ή υπερκλάσεις) στις παραμέτρους.(Μια κλάση που παράγεται από μια άλλη κλάση καλείται **υποκλάση** (subclass) ή παραγόμενη κλάση (derived) ή θυγατρική κλάση (child). Η κλάση από την οποία παράγεται μια κλάση λέγεται **υπερκλάση** (superclass) ή γονική κλάση (parent))
- Παράδειγμα μετασχηματισμού της μεθόδου totalArea

```
public static double totalArea (Collection<? extends Shape> arr) {
      double total = 0.0;
      for ( Shape s : arr ) //For-Each Shape s in arr
                if ( s != null ) total += s.area();
      return total;
}
```

- Χωρίς τη χρήση WildCard: **`Collection<T extends Shape>`**

Έλεγχος Ορίων Τύπων (Type Bounds)

- Υποθέστε την πιο κάτω μέθοδο

```
public static <AnyType> AnyType findMax( AnyType [ ] arr ) {
        int maxIndex = 0;
        for( int i = 1; i < arr.length; i++ )
                if( arr[ i ].compareTo( arr[ maxIndex ] ) > 0 )
                        maxIndex = i;
        return arr[ maxIndex ];
}
```

- Η πιο κάτω γραμμή
  `arr[i].compareTo( arr[maxIndex] ) > 0`
  προϋποθέτει την υλοποίηση της **διαπροσωπείας Comparable**

- **<u>Ερώτηση:</u>** Έχουμε ορίσει ότι ο τύπος AnyType υλοποιεί την διαπροσωπεία Comparable; ➔ ΟΧΙ (σφάλμα μεταγλώττισης)

Χρήση Type Bounds

- Πρέπει με κάποιο τρόπο να αποδεχόμαστε μόνο αντικείμενα που υποστηρίζουν την μέθοδο compareTo (δηλ., να υλοποιούν την διαπροσωπεία Comparable)
- Υποθέστε την πιο κάτω εναλλακτική υλοποίηση:

```
public static <AnyType extends Comparable<AnyType>>
    AnyType findMax( AnyType[] arr ) {
    int maxIndex = 0;
    for( int i = 1; i < arr.length; i++ )
          if( arr[i].compareTo( arr[ maxIndex ] ) > 0 )
                maxIndex = i;
    return arr[ maxIndex ];
}
```

- Ο πιο πάνω κώδικας μεταγλωττίζεται
- Η γραμμή κώδικα **<AnyType extends Comparable<AnyType>>** είναι ένα type bound

Χρήση Type Bounds (συν.)

- …συνέχεια από την προηγούμενη διαφάνεια
- Υποθέστε ότι η κλάση Shape υλοποιεί την διαπροσωπεία Comparable<Shape>
- Υποθέστε επίσης ότι η κλάση Circle κληρονομεί από την Shape
- Συμπέρασμα:
  - Η κλάση Square ΕΙΝΑΙ ένα Comparable<Shape>
  - Η κλάση Square ΔΕΝ ΕΙΝΑΙ ένα Comparable<Square>

- Συνεπώς πρέπει με κάποιο τρόπο να πούμε ότι AnyType ΕΊΝΑΙ ένα Comparable<T> όπου T είναι μία υπερκλάση του AnyType

- Για να το καταφέρουμε, μπορούμε να χρησιμοποιήσουμε ένα wildcard

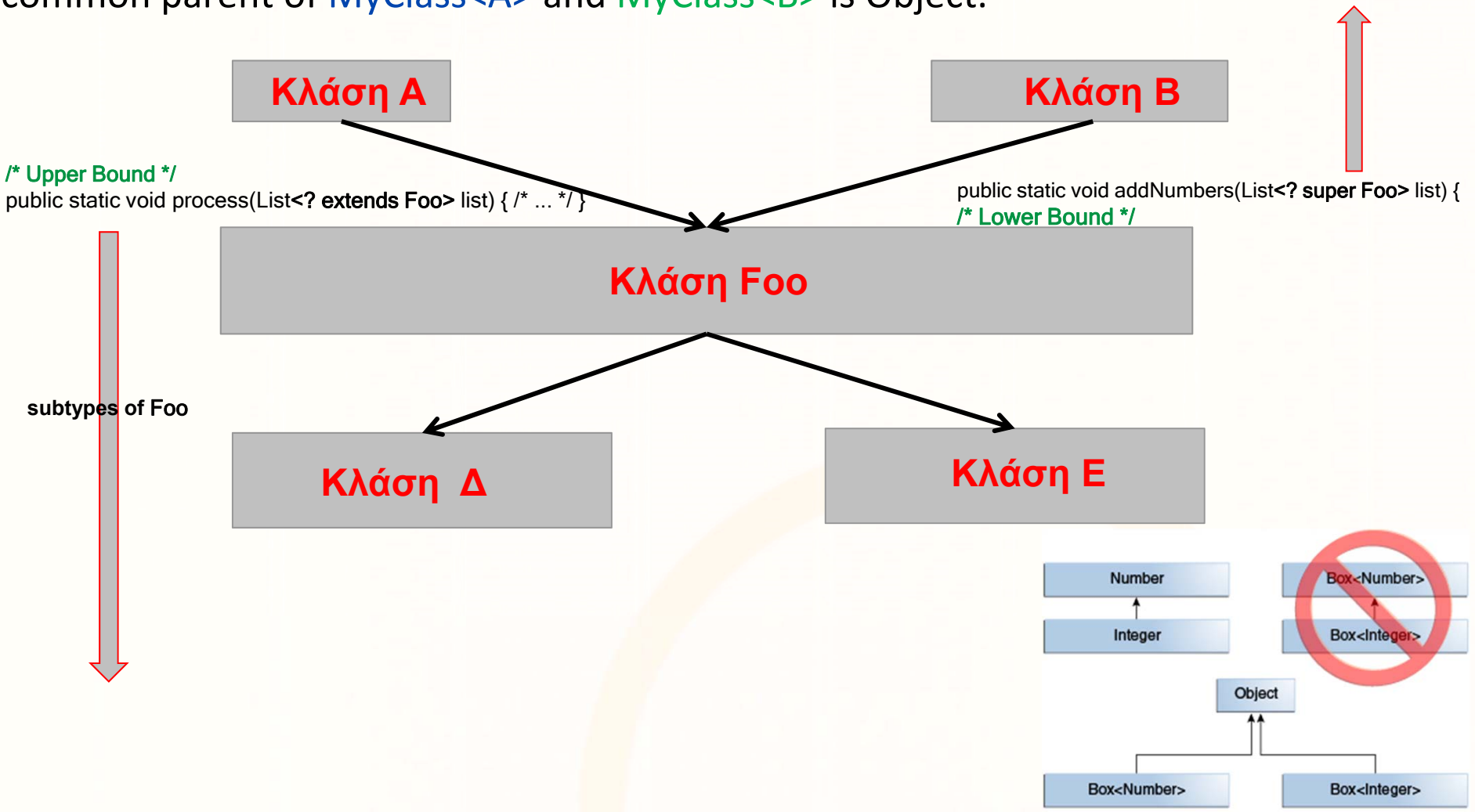
```
public static <AnyType extends
                        Comparable<? super AnyType>>
                    AnyType findMax( AnyType [ ] arr ) {
        int maxIndex = 0;
        for( int i = 1; i < arr.length; i++ )
                if( arr[ i ].compareTo( arr[ maxIndex ] ) > 0 )
                        maxIndex = i;
        return arr[ maxIndex ];
}
```

- Ο μεταγλωττιστής θα δέχεται πίνακες από αντικείμενα τύπου AnyType, έτσι ώστε το T υλοποιεί την διαπροσωπεία Comparable<S>, όπου το T ΕΙΝΑΙ S.

- Ευτυχώς, στο μάθημα δεν θα δούμε τίποτα πιο περίπλοκο από το πιο πάνω!

Κληρονομικότητα Ορολογία

Given two concrete types A and B (for example, Number and Integer), MyClass<A> has no relationship to MyClass<B>, regardless of whether or not A and B are related. The common parent of MyClass<A> and MyClass<B> is Object.



References

- Java Generic types http://docs.oracle.com/javase/tutorial/java/generics/index.html
- Mark A. Weiss - Data Structures and Algorithm Analysis in Java (3rd Edition) - Publication Date: November 28, 2011 ISBN-10: 0132576279
- ISBN-13: 978-0132576277
- http://en.wikipedia.org/wiki/Generics_in_Java
- http://docs.oracle

Άσκηση 1

- Υλοποιήστε μία κλάση MemoryCell ως εξής:
  - Να έχει μία private μεταβλητή storedValue τυχαίου τύπου
  - Ένα άδειο κατασκευαστή
  - Ένα κατασκευαστή που να δέχεται ένα αντικείμενο τυχαίου τύπου
  - Μία μέθοδο read() που επιστρέφει την μεταβλητή storedValue
  - Μία μέθοδο write() που να ενημερώνει την μεταβλητή storedValue

- Δημιουργείστε ένα πρόγραμμα που να ελέγχει την πιο πάνω κλάση

java.lang

Interface Comparable<T>

Type Parameters:

T - the type of objects that this object may be compared to

All Known Subinterfaces:

Delayed, Name, Path, RunnableScheduledFuture<V>, ScheduledFuture<V>

All Known Implementing Classes:

AbstractRegionPainter.PaintContext.CacheMode, AccessMode, AclEntryFlag, AclEntryPermission, AclEntryType, AddressingFeature.Responses, Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, CertPathValidatorException.BasicReason, Character, Character.UnicodeScript, CharBuffer, Charset, ClientInfoStatus, CollationKey, Component.BaselineResizeBehavior, CompositeName, CompoundName, CRLReason, CryptoPrimitive, Date, Date, Desktop.Action, Diagnostic.Kind, Dialog.ModalExclusionType, Dialog.ModalityType, Double, DoubleBuffer, DropMode, ElementKind, ElementType, Enum, File, FileTime, FileVisitOption, FileVisitResult, Float, FloatBuffer, Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, GraphicsDevice.WindowTranslucency, GregorianCalendar, GroupLayout.Alignment, IntBuffer, Integer, JavaFileObject.Kind, JTable.PrintMode, KeyRep.Type, LayoutStyle.ComponentPlacement, LdapName, LinkOption, Locale.Category, Long, LongBuffer, MappedByteBuffer, MemoryType, MessageContext.Scope, Modifier, MultipleGradientPaint.ColorSpaceType, MultipleGradientPaint.CycleMethod, NestingKind, Normalizer.Form, NumericShaper.Range, ObjectName, ObjectStreamField, PKIXReason, PosixFilePermission, ProcessBuilder.Redirect.Type, Proxy.Type, PseudoColumnUsage, Rdn, Resource.AuthenticationType, RetentionPolicy, RoundingMode, RowFilter.ComparisonType, RowIdLifetime, RowSorterEvent.Type, Service.Mode, Short, ShortBuffer, SOAPBinding.ParameterStyle, SOAPBinding.Style, SOAPBinding.Use, SortOrder, SourceVersion, SSLEngineResult.HandshakeStatus, SSLEngineResult.Status, StandardCopyOption, StandardLocation, StandardOpenOption, StandardProtocolFamily, String, SwingWorker.StateValue, Thread.State, Time, Timestamp, TimeUnit, TrayIcon.MessageType, TypeKind, URI, UUID, WebParam.Mode, Window.Type, XmlAccessOrder, XmlAccessType, XmlNsForm

---

```
public interface Comparable<T>
```

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's `compareTo` method is referred to as its *natural comparison method*.

Lists (and arrays) of objects that implement this interface can be sorted automatically by `Collections.sort` (and `Arrays.sort`). Objects that implement this interface can be used as keys in a sorted map or as elements in a sorted set, without the need to specify a comparator.

Method Detail

compareTo

`int compareTo(T o)`

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive.

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Throws:

`NullPointerException` - if the specified object is null

`ClassCastException` - if the specified object's type prevents it from being compared to this object.