Ενότητα 2: **Διαχείριση Μνήμης** και Σχεδιασμός Κλάσεων

Μάθημα 13: 5/3/2024

Αναλλοίωτοι Περιορισμοί

University of Cyprus
Department of Computer Science

Τι εκτυπώνουν οι εντολές:

```
System.out.println(1 + 2 + "c");
System.out.println("c" + 1 + 2);
```

- **Συντακτικό σφάλμα**
- **3c** και **c3**
- **12c** και **c12**
- **3c** και **c12**

00:30

00:30

Ποιο είναι το αποτέλεσμα των ακόλουθων εκφράσεων;

```
1: int i = 8L;
2: long j = (int) 8_500.29 + (short) 15000;
```

- **Συντακτικό σφάλμα και στις δύο.**
- **Συντακτικό σφάλμα στην (2).**
- **Συντακτικό σφάλμα στην (1). Η δεύτερη δίνει στην j την τιμή 23500.**
- **Το i παίρνει τιμή 8 και το j 15008**

00:30

00:30

Περίγραμμα

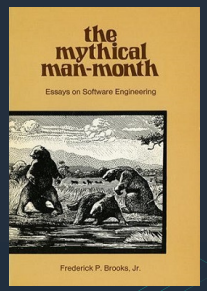
University of Cyprus
Department of Computer Science

Ενότητα 2: Διαχείριση Μνήμης και **Σχεδιασμός Κλάσεων**

Αναλλοίωτοι Περιορισμοί

University of Cyprus
Department of Computer Science

Δομές Δεδομένων



University of Cyprus
Department of Computer Science

Δείξε μου τα διαγράμματα ροής σου και κρύψε τους πίνακες σου και θα συνεχίσω να παραξενεύομαι.

Δείξτε μου τους πίνακες σου και δεν θα χρειαστώ συνήθως τα διαγράμματα ροής. θα είναι προφανή.

Frederick P. Brooks, Jr., The Mythical Man Month

ΒΡΕΙΤΕ ΤΟ ΠΡΟΒΛΗΜΑ

```
/**
 * A class of car objects that can move forward, backward
 * and turn!
 * @author mdd
 *
 */
public class Car {
    private int odometer = 0;
    priva

    publi

    publi
    }

    publi

        System.out.println(owner +  s car nas moved   + dist +   units. );
    }

    public void turnCar(double angle) {
        System.out.println(owner + "'s car has turned " + angle + "
units.");
    }

    public int getOdometer() {
        return odometer;
    }
```

```
public static void main(String[] args) {
```

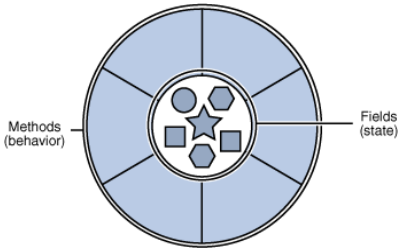
ΤΟ ΟΔΟΜΕΤΡΟ ΔΕΝ ΜΠΟΡΕΙ ΝΑ ΛΑΒΕΙ ΑΡΝΗΤΙΚΗ ΤΙΜΗ!!!

00:30

Σχεδιασμός κλάσης Person

University of Cyprus
Department of Computer Science

- Ποια πεδία δεδομένων;
- Με ποιες τιμές;
- Ποιες μέθοδοι;
- (Class or Type) Invariant = Αναλλοίωτος - αμετάβλητος περιορισμός



Μ. Δικαιάκος, ΕΠΛ133

Σχεδιασμός κλάσης Person: Πεδία δεδομένων

- Μια απλή κλάση **`Person`** θα μπορούσε να περιέχει μεταβλητές που αντιπροσωπεύουν το όνομα ενός ατόμου, την ημερομηνία που γεννήθηκε και την ημερομηνία που πέθανε.
- Αυτές οι μεταβλητές στιγμιότυπου θα είναι όλες τύποι κλάσης:
  - **`name`** - τύπου **`String`**
  - ημερομηνίες **`born`**, **`died`** - τύπου **`Date`**
- Καθεμία από τις μεταβλητές του στιγμιότυπου θα δηλωθεί ως **`private`**

```
public class Person {
  private String name;
  private Date born;
  private Date died;  //null is still alive
  . . .
```


Κλάση Date

University of Cyprus
Department of Computer Science

```
import java.util.Scanner;

public class Date
{
    private String month;
    private int day;
    private int year; //a four digit number.

    public Date( )
    {
        month = "January";
        day = 1;
        year = 1000;
    }

    public Date(int monthInt, int day, int year) { setDate(monthInt, day, year); }

    public Date(String monthString, int day, int year) { setDate(monthString, day, year); }

    public Date(int year) { setDate( monthInt: 1,  day: 1, year); }

    public Date(Date aDate)
    {
        if (aDate == null)//Not a real date.
        {
            System.out.println("Fatal Error.");
            System.exit( status: 0);
        }

        month = aDate.month;
        day = aDate.day;
        year = aDate.year;
    }
```

Κλάση Date

University of Cyprus
Department of Computer Science

```
public boolean precedes(Date otherDate)
{
    return ( (year < otherDate.year) ||
       (year == otherDate.year && getMonth( ) < otherDate.getMonth( )) ||
       (year == otherDate.year && month.equals(otherDate.month)
                                  && day < otherDate.day) );
}
```


Σχεδιασμός Person: κατασκευαστής

- Για να υπάρχει κάποιος άνθρωπος πρέπει να έχει (τουλάχιστον) όνομα και ημερομηνία γέννησης
  - Έχει νόημα να δώσουμε κατασκευαστή κλάσης **`Person`** χωρίς τυπικές παραμέτρους;
- Ένα άτομο που είναι ακόμα ζωντανό δεν έχει ακόμη ημερομηνία θανάτου
  - Επομένως, ο κατασκευαστής **`Person`** θα πρέπει να μπορεί να έχει τιμή **`null`** για την ημερομηνία θανάτου
- Ένα άτομο που έχει πεθάνει πρέπει να έχει ημερομηνία γέννησης που προηγήθηκε της ημερομηνίας θανάτου του
  - Επομένως, όταν παρέχονται και οι δύο ημερομηνίες, θα πρέπει να **ελεγχθούν** για συνέπεια (έλεγχος συνέπειας)
  - Που;

Αναλλοίωτος Περιορισμός Κλάσης

- Για την κλάση `Person`, τα ακόλουθα πρέπει να ισχύουν πάντοτε σε όλα τα αντικείμενα `Person`:

Κάθε αντικείμενο της κλάσης `Person` πρέπει να έχει ημερομηνία γέννησης (που δεν είναι null).

Αν το αντικείμενο έχει ημερομηνία θανάτου, τότε αυτή θα πρέπει να είναι ίση ή μεγαλύτερη από την ημερομηνία γέννησης.

- Οι κατασκευαστές και μέθοδοι της `Person` πρέπει να διασφαλίζουν την ισχύ του πιο πάνω περιορισμού.

Αναλλοίωτοι Περιορισμοί (Class Invariants)

- Invariant = Αναλλοίωτος - αμετάβλητος περιορισμός
- Ο αναλλοίωτος περιορισμός κλάσης (**class or type invariant**) είναι μια συνθήκη που πρέπει να παραμένει εν ισχύι στη διάρκεια της ζωής κάθε αντικειμένου της κλάσης.

Αναλλοίωτοι Περιορισμοί (Class Invariants)

- Οι αναλλοίωτοι περιορισμοί επιβάλλονται κατά την κατασκευή αντικειμένων και διατηρούνται συνεχώς μεταξύ κλήσεων προς δημόσιες μεθόδους αυτών.
- **Προσωρινή παραβίαση** ενός αναλλοίωτου περιορισμού *μεταξύ διαδοχικών κλήσεων σε ιδιωτικές μεθόδους* είναι δυνατή, αν και δεν ενθαρρύνεται.
- Ο ορισμός των invariants μπορεί να βοηθήσει τους προγραμματιστές και τους δοκιμαστές να εντοπίσουν περισσότερα σφάλματα κατά τη διάρκεια της δοκιμής του λογισμικού.

Κατασκευαστής `Person`

```
public Person(String initialName, Date birthDate, Date deathDate){
  if (consistent(birthDate, deathDate)){
    name = initialName;
    born = birthDate;
    if (deathDate == null)
      died = null;
    else
      died = deathDate;
  }
  else   {
    System.out.println("Inconsistent dates.");
    System.exit(0);
  }
}
```


Αναλλοίωτος Περιορισμός Κλάσης

```
/**  Class invariant: A Person always has a date of birth,
     and if the Person has a date of death, then the date of
     death is equal to or later than the date of birth.
     To be consistent, birthDate must not be null. If there
     is no date of death (deathDate == null), that is
     consistent with any birthDate. Otherwise, the birthDate
     must come before or be equal to the deathDate.
*/
private static boolean consistent(Date birthDate,
                                  Date deathDate){
    if (birthDate == null)
       return false;
    else if (deathDate == null)
       return true;
    else  return (birthDate.precedes(deathDate) ||
                  birthDate.equals(deathDate));
 }
```



```
public boolean precedes(Date otherDate) {
 return ((year < otherDate.year) ||
         (year == otherDate.year && getMonth() < otherDate.getMonth())||
         (year == otherDate.year && month.equals(otherDate.month)
                        && day < otherDate.day));
    }
```

```
public boolean equals(Date otherDate) {
        if (otherDate == null)
            return false;
        else
            return ((month.equals(otherDate.month)) &&
                    (day == otherDate.day) && (year == otherDate.year));
    }
```


00:30

Πώς συγκρίνουμε δύο αντικείμενα Person για να δούμε αν αφορούν στο ίδιο άτομο;

```
public class Person {
  private String name;
  private Date born;
  private Date died;//null is alive
  . . .
```

University of Cyprus
Department of Computer Science

Σχεδιασμός Κλάσης `Person`: μέθοδος `equals`

```
public boolean equals(Person otherPerson){
  if (otherPerson == null)
    return false;
  else
    return (name.equals(otherPerson.name) &&
            born.equals(otherPerson.born) &&
            died.equals(otherPerson.died));
}
```


Κλάση Person: `equals` και `datesMatch`

- Ο ορισμός της **`equals`** για την κλάση **`Person`** περιλαμβάνει μια κλήση **`equals`** για την κλάση **`String`** και μια κλήση **`equals`** για την κλάση Date.
- Η Java καθορίζει ποια ακριβώς μέθοδος **`equals`** θα κληθεί από τον τύπο του καλούντος αντικειμένου της.

Σχεδιασμός Κλάσης `Person`: μέθοδος `equals`

```
public boolean equals(Person otherPerson){
  if (otherPerson == null)
    return false;
  else
    return (name.equals(otherPerson.name) &&
            born.equals(otherPerson.born) &&
            died.equals(otherPerson.died));
}
```

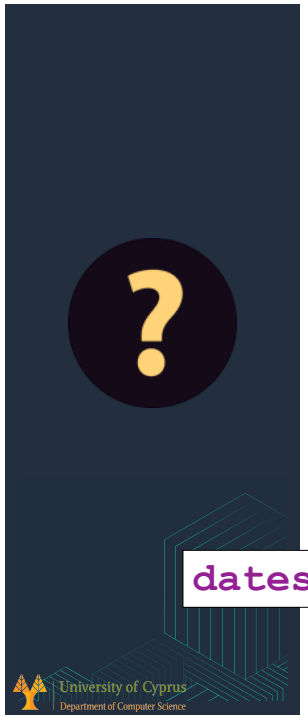
ΒΡΕΙΤΕ ΤΟ ΠΡΟΒΛΗΜΑ

Σχεδιασμός Κλάσης `Person`: μέθοδος `equals`

```
public boolean equals(Person otherPerson){
  if (otherPerson == null)
    return false;
  else
    return (name.equals(otherPerson.name) &&
            born.equals(otherPerson.born) &&
            died.equals(otherPerson.died));
}
```


Σχεδιασμός Κλάσης `Person`: μέθοδος `equals`

```
public boolean equals(Person otherPerson){
  if (otherPerson == null)
    return false;
  else
    return (name.equals(otherPerson.name) &&
            born.equals(otherPerson.born) &&
            datesMatch(died, otherPerson.died));
}
```



Οι μεταβλητές στιγμιοτύπου **died** συγκρίνονται με την **datesMatch** αντί της **equals**.

Γιατί;

```
datesMatch(died, otherPerson.died)
```


Σχεδιασμός Person: η μέθοδος datesMatch

```
/**  To match date1 and date2 must either be the
     same date or both be null.
*/
private static boolean datesMatch(Date date1,
                                  Date date2){
  if (date1 == null)
    return (date2 == null);
  else if (date2 == null) //&& date1 != null
    return false;
  else // both dates are not null.
    return(date1.equals(date2));
}
```


Σχεδιασμός Person: η μέθοδος toString

- Η **toString** της **Person** καλεί την **toString** της **Date**:

```
public String toString( ) {
  String diedString;
  if (died == null)
    diedString = ""; //Empty string
  else
    diedString = died.toString( );

  return (name + ", " + born + "-" +
 diedString);
}
```


Αναλλοίωτοι Περιορισμοί

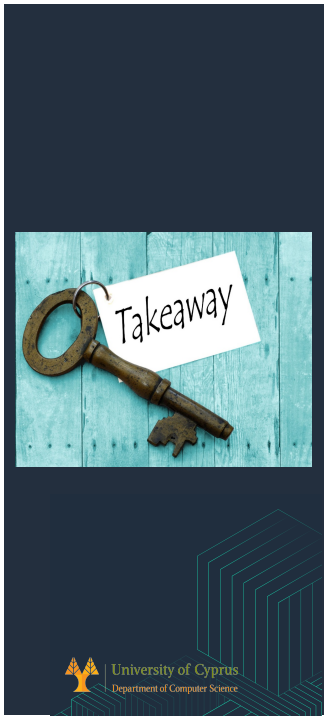
Παραβίαση ιδιωτικότητας (Privacy leaks)

Using and Misusing References

- Είναι σημαντικό να διασφαλίζουμε στα προγράμματά μας ότι οι **ιδιωτικές** μεταβλητές στιγμιοτύπου (private instance variables) παραμένουν **πραγματικά ιδιωτικές**!
- Διαρροή/παραβίαση ιδιωτικότητας μια κλάσης Α συμβαίνει όταν:

  *Μέλος τρίτης κλάσης μπορεί να αποκτήσει πρόσβαση σε **ιδιωτικό πεδίο** της Α.*

- Πώς;;
- Για μια μεταβλητή στιγμιοτύπου **αρχέγονου τύπου** (primitive type instance variable), η δήλωσή της ως `private` αρκεί για να διασφαλίσει ότι δεν θα υπάρξουν διαρροές ιδιωτικότητας privacy leaks (*παραβίαση ιδιωτικότητας-διαρροή*).



Για μεταβλητές στιγμιοτύπου **τύπου κλάσης** (class type instance variable)

η δήλωσή τους ως `private`

δεν αρκεί για την αποφυγή διαρροής


```
public class Date {
    private String month;
    private int day;
    private int year; //a four digit number.

    public void setMonth(int monthNum) { }

   }
```

```
public class Person {
  private String name;
  private Date born;
  private Date died; //if null still alive
  Person(String name,
         Date bdate, Date ddate) {
       this.name = name;
       this.born = bdate;
       this.died = ddate;
       }

        …
 }
```



```
public class Date {
    private String month;
    private int day;
    private int year; //a four digit number.

    public void setMonth(int monthNum) { }

   }
```

```
public class Person {
  Person(String name,
         Date bdate, Date ddate) {
        this.name = name;
        this.born = bdate;
        this.died = ddate;
        }
  private String name;
  private Date born;
  private Date died; //if null still alive
      …
}
```

```
public class BabyFactory {

   public static Person[] babyDelivery(String[] names, Date today) {
     if (names != null) {
       Person[] newBornBabies = new Person[names.length];
       for (i=0; i<names.length; i++}
         newBornBabies[i] = new Person(names[i], today, null);
       return newBornBabies;
     }
    else return null;
   } … }
```

Copy Constructor for a Class with Class Type Instance Variables

- Η κλάση **Person** περιλαμβάνει τρεις μεταβλητές στιγμιοτύπου (1 String και 2 Person)
- Αν οι μεταβλητές **born** και **died** για το νέο αντικείμενο **Person** αντιγραφούν απλά από τα ορίσματα του κατασκευαστή:

```
born = bdate; //dangerous
died = ddate; //dangerous
```

BAD PRACTICE

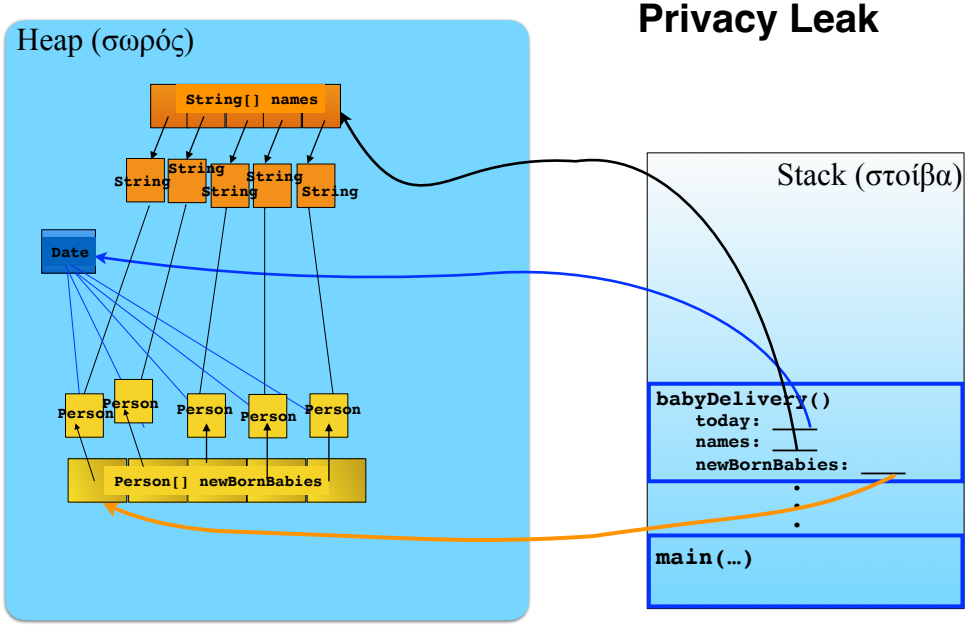
- δεν θα δημιουργηθούν ανεξάρτητα αντίγραφα των αρχικών αντικειμένων `Date` αλλά
- … αντίγραφα (ψευδώνυμα-aliases) των μεταβλητών **bdate** και **ddate**

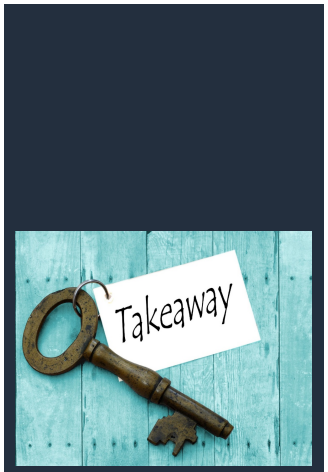

```
public class Date {
    private String month;
    private int day;
    private int year; //a four digit number.

    public void setMonth(int monthNum) { }

}
```

```
public class Person {
  Person(String name,
         Date bdate, Date ddate) {
         this.name = name;
         this.born = bdate;
         this.died = ddate;
         }
  private String name;
  private Date born;
  private Date died; //if null still alive
     …
}
```

```
public class BabyFactory {

   public static Person[] babyDelivery(String[] names, Date today) {
     if (names != null) {
       Person[] newBornBabies = new Person[names.length];
       for (i=0; i<names.length; i++}
         newBornBabies[i] = new Person(names[i], today, null);
       return newBornBabies;
     }
    else return null;
   } … }
```

Privacy Leak





Οποιαδήποτε κλάση έχει πρόσβαση σε κάποιο αντικείμενο Date, μπορεί να αλλάξει και να επηρεάσει τις ημερομηνίες γέννησης όλων των αντικειμένων Person, τα πεδία δεδομένων των οποίων έχουν στο αντικείμενο Date

Αναλλοίωτοι Περιορισμοί

Κατασκευαστές αντιγράφου (Copy constructors)

University of Cyprus
Department of Computer Science

Copy Constructors - Κατασκευαστές αντιγράφου

- Ένας «κατασκευαστής αντιγράφου» (copy constructor) είναι ένας κατασκευαστής που:
- Δέχεται **ένα μόνο όρισμα**, ίδιου τύπου με την κλάση του.
- Δημιουργεί ένα αντικείμενο που είναι ξεχωριστό και ανεξάρτητο από το όρισμα, αλλά με τιμές μεταβλητών στιγμιότυπων ορισμένες ώστε να πρόκειται για *ακριβές αντίγραφο του αντικειμένου του ορίσματος*.

ΤΙ ΜΠΟΡΟΥΜΕ ΝΑ ΚΑΝΟΥΜΕ ΓΙΑ ΑΠΟΦΥΓΗ ΤΗΣ ΔΙΑΡΡΟΗΣ ΙΔΙΩΤΙΚΟΤΗΤΑΣ;

University of Cyprus
Department of Computer Science

M. D. Dikaiakos

Κατασκευαστής Αντιγράφου κλάσης με πεδία δεδομένων τύπου κλάσης

- Ο κατασκευαστής αντιγράφου για την κλάση `Person` καθίσταται «ασφαλής» αν κατασκευάσει νέα και ανεξάρτητα αντίγραφα των `born` και `died`, και συνεπώς, ένα εντελώς νέο και ανεξάρτητο αντίγραφο του αρχικού αντικειμένου `Person`:

```
born = new Date(bdate);
died = new Date(ddate);
```

- Για να συμβεί αυτό, όταν η κλάση μας έχει μεταβλητές στιγμιοτύπου τύπου κλάσης (class type instance variables):
  - *πρέπει να έχουν ήδη οριστεί κατασκευαστές αντιγραφείς για τις κλάσεις των μεταβλητών στιγμιοτύπου.*

Κατασκευαστής Αντιγράφου κλάσης με πεδία δεδομένων αρχέγονου τύπου

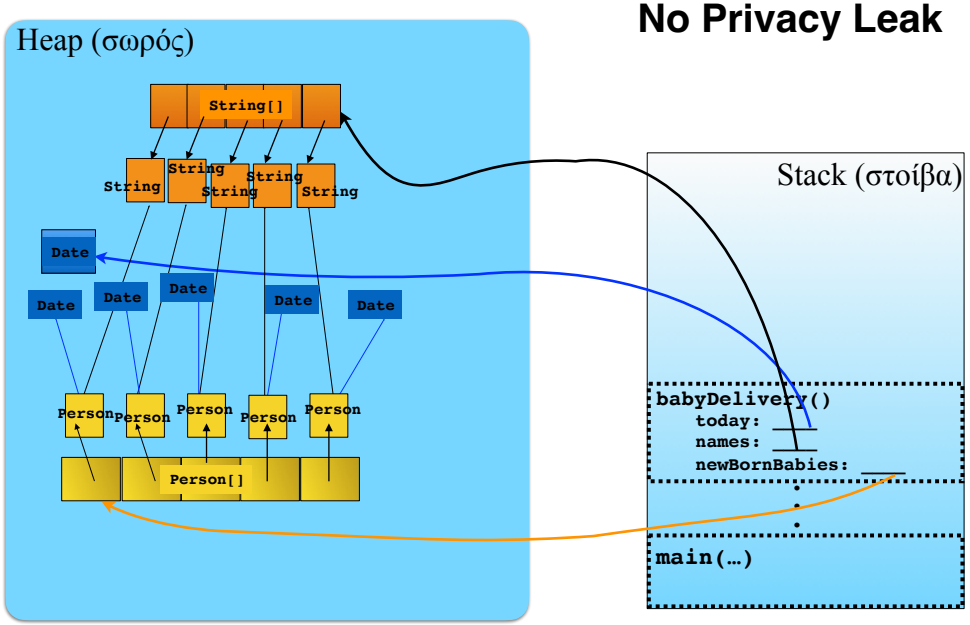
- Στον κατασκευαστή-αντιγράφου της `Date`, οι τιμές όλων των μεταβλητών ιδιωτικού στιγμιότυπου αρχέγονου τύπου απλώς αντιγράφονται:

```
public Date(Date aDate) {
  if (aDate == null)  { //Not a real date.
    System.out.println("Fatal Error.");
    System.exit(0);
  }
  month = aDate.month;
  day = aDate.day;
  year = aDate.year;
}
```


Κατασκευαστής Αντιγράφου κλάσης με πεδία δεδομένων τύπου κλάσης

```
public Person(Person original) {
  if (original == null)  {
    System.out.println("Fatal error.");
    System.exit(0);
  }
  name = original.name;
  born = new Date(original.born);
  if (original.died == null)
    died = null;
  else
    died = new Date(original.died);
}
```


No Privacy Leak



Προσοχή! Διαρροή ιδιωτικότητας

- Στο προηγούμενο παράδειγμα είδαμε ότι ο *εσφαλμένος ορισμός του κατασκευαστή* της κλάσης `Person` οδήγησε σε διαρροή ιδιωτικότητας (*privacy leak*).
- Αντίστοιχο πρόβλημα μπορεί να προκύψει από εσφαλμένο ορισμό μέθοδου μεταλλαγής ή μεθόδου προσπέλασης
  - Για παράδειγμα:

```
public Date getBirthDate(){
    return born;  //dangerous
}
```

  - Instead of:

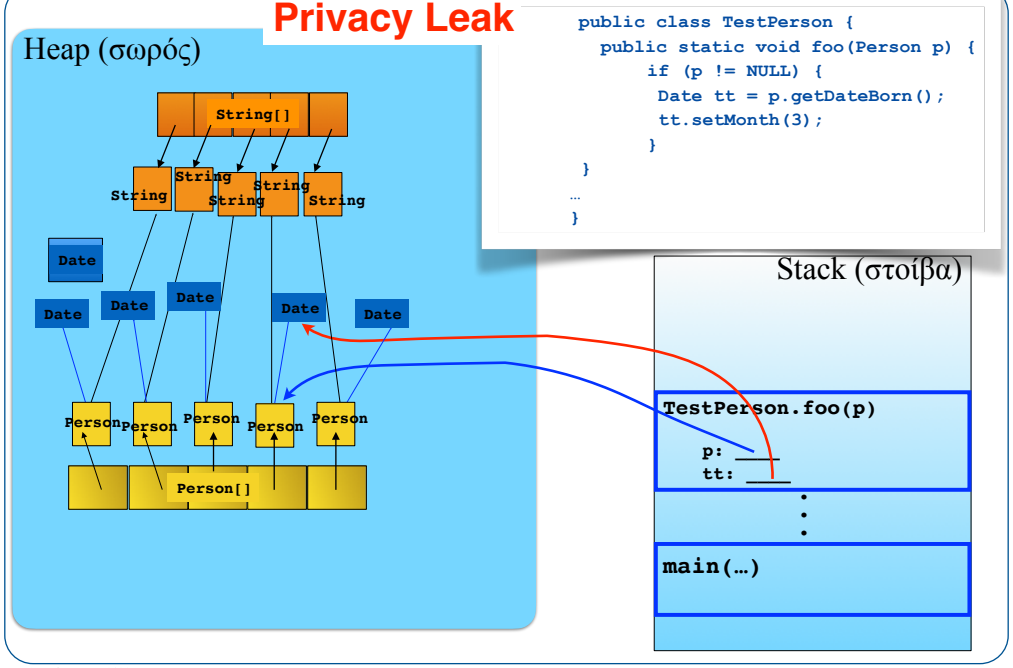
```
public Date getBirthDate(){
    return new Date(born);  //correct
}
```

Privacy Leak

```
public class Person {
  Person(String name,
         Date bdate, Date ddate) {
        this.name = name;
        this.born = new Date(bdate);
        this.died = new Date(ddate);
        }
  private String name;
  private Date born;
  private Date died; //if null still alive
  . . .
  public Date getDateBorn() {
       return born;
  }
…
}
```

```
public class TestPerson {
  public static void foo(Person p) {
       if (p != NULL) {
        Date tt = p.getDateBorn();
        tt.setMonth(3);
       }
 }
…
}
```


Privacy Leak




```
public class TestPerson {
  public static void foo(Person p) {
       if (p != NULL) {
        Date tt = p.getDateBorn();
        tt.setMonth(3);
       }
 }
…
}
```


Μεταλλάξιμες και μη μεταλλάξιμες κλάσεις (Mutable and Immutable Classes)

- Η μέθοδος προσπέλασης **`getName`** της κλάσης **`Person`** εμφανίζεται να αντιβαίνει στους κανόνες για αποφυγή διαρροών ιδιωτικότητας:

```
public String getName(){
   return name; //Isn't this dangerous?
}
```

- Αν και φαίνεται η ίδια περίπτωση με τα αντιπαραδείγματα που είδαμε πριν, στην πραγματικότητα δεν είναι:
  - Η κλάση **`String`** **δεν περιλαμβάνει μεθόδους μεταλλαγής** που να μπορούν να αλλάξουν τα δεδομένα ενός αντικειμένου **`String`**

Μεταλλάξιμες και μη μεταλλάξιμες κλάσεις (Mutable and Immutable Classes)

- Μια κλάση που δεν περιέχει μεθόδους (εκτός από τους κατασκευαστές) οι οποίες αλλάζουν οποιοδήποτε από τα δεδομένα σε ένα αντικείμενο της κλάσης ονομάζεται *immutable class (μη μεταλλάξιμη κλάση)*
  - Αντικείμενα τέτοιων κλάσεων αποκαλούνται μη μεταλλάξιμα/αμετάβλητα αντικείμενα (*immutable objects*)
  - Είναι απολύτως ασφαλές να επιστρέψετε μια αναφορά σε ένα αμετάβλητο αντικείμενο επειδή το αντικείμενο δεν μπορεί να αλλάξει με κανέναν τρόπο
  - The κλάση **String** είναι αμετάβλητη/μη μεταλλάξιμη κλάση

Μεταλλάξιμες και μη μεταλλάξιμες κλάσεις

- Μια κλάση που περιέχει δημόσιες μεθόδους μεταλλαγής ή άλλες δημόσιες μεθόδους που μπορούν να αλλάξουν τα πεδία δεδομένων των αντικειμένων της ονομάζεται *μεταλλάξιμη κλάση* (*mutable class*) και τα αντικείμενα της καλούνται *μεταλλάξιμα αντικείμενα* (*mutable objects*)
- **Never write a method that returns a mutable object!**
- Instead, use a copy constructor to return a reference to a completely independent copy of the mutable object

Deep Copy Versus Shallow Copy

- A *deep copy* (βαθύ αντίγραφο) of an object is a copy that has no references in common with the original
  - One Exception: References to immutable objects are allowed to be shared
- Any copy that is not a deep copy is called a *shallow copy* (επιφανειακό αντίγραφο).
  - This type of copy can cause dangerous privacy leaks in a program

Κατασκευαστής `Person`

```
public Person(String initialName, Date birthDate, Date deathDate){
  if (consistent(birthDate, deathDate)){
    name = initialName;
    born = new Date(birthDate);
    if (deathDate == null)
      died = null;
    else
      died = new Date(deathDate);
  }
  else   {
    System.out.println("Inconsistent dates.");
    System.exit(0);
  }
}
```

