



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

برنامه سازی پیشرفته و کارگاه

Encapsulation (کیسوله سازی)

استاد درس

دکتر مهدی قطعی

استاد دوم

بهنام یوسفی مهر

نگارش

ایلیا اسدی

بهار ۱۴۰۳

فهرست

| | |
|----|---|
| 3 | مقدمه..... |
| 4 | چرا از Encapsulation استفاده می‌کنیم؟..... |
| 5 | چطوری توی کدمون Encapsulation رو به کار ببریم؟..... |
| 5 | کلیدواژه‌ی private..... |
| 7 | getter..... |
| 9 | setter..... |
| 10 | کلیدواژه‌ی protected..... |
| 12 | package-private..... |
| 15 | چه چیزی یاد گرفتیم؟..... |

مقدمه

کپسوله سازی (Encapsulation) یکی دیگه از اصول پایه ای شی گرایی هستش. encapsulation بهتون این قابلیت رو می ده که جزئیات پیاده سازی های شما، یعنی فیلدها، متدها و کلاس ها، برای کلاس های دیگه اصطلاحاً hide بشه و دسترسی مستقیم وجود نداشته باشه. به عبارتی، encapsulation دیتا و کدی که برای اون نوشتین رو به عنوان یک واحد در نظر می گیره که این واحد، همون کلاسی هست که داخلش دارین این ها رو تعریف می کنین و یکی از feature هایی که داره ، hide شدن اون جزئیات کد تونه.

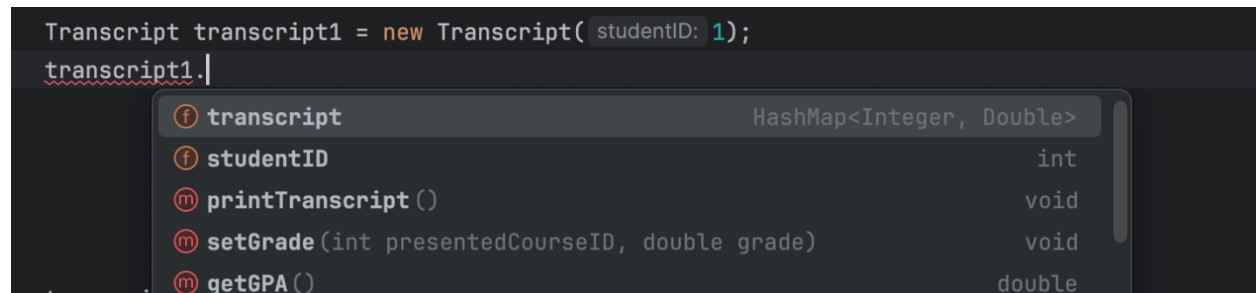
توی این داکيومنت، با اصل encapsulation و متدهای getter و setter کامل آشنا می شین و آخر سر هم، یه تمرین کارگاه برای encapsulation داریم!

چرا از Encapsulation استفاده می کنیم؟

بذارین با یه مثال شروع کنیم. توی پروژه‌ی شبیه ساز دانشگاه، شما قبل فیلدها و متدهایی که تعریف میکردین، کلید واژه‌ی public رو قرار می دادین. مثلاً داخل کلاس Transcript، فیلدها رو به این صورت تعریف میکردین:

```
public class Transcript {
    public int studentID;
    public HashMap<Integer, Double> transcript = new HashMap<>();
}
```

این به این معنی هست که وقتی داخل یک کلاس دیگه، مثلاً Main، آبجکتی از کلاس Transcript بسازین، میتونین بدون مشکل، تنها با یک dot notation، به فیلد studentID دسترسی داشته باشین:



The screenshot shows a code editor with the following code: `Transcript transcript1 = new Transcript(studentID: 1);` and `transcript1.` Below the code, a dropdown menu displays the members of the `Transcript` class: `transcript` (type `HashMap<Integer, Double>`), `studentID` (type `int`), `printTranscript()` (type `void`), `setGrade(int presentedCourseID, double grade)` (type `void`), and `getGPA()` (type `double`).

به همین صورت برای متدها هم همین قضیه برقراره. اما اینطوری تعریف کردن فیلدهامون، اشکالاتی داره. با یه مثال این مشکل رو بررسی می کنیم.

مثلاً تصور کنین شما توی Main، برای یه دانشجو با یه studentID مشخص، یه کارنامه تعریف کردین. حالا، میتونین راحت اون شماره‌ی دانشجویی رو عوض کنین و با این کار، این کارنامه تبدیل به کارنامه‌ی یه شخص دیگه میشه یا اصلاً یه شماره‌ی دانشجویی ناموجود رو براش قرار میدین! پس، یعنی اولاً که کنترل کمتری روی داده‌ها و متدها دارین خصوصاً اگر بخواین محدودیتی (مثلاً مقداری که میتونه اون فیلد بگیره) روی فیلدهاتون بذارین. دوماً که دسترسی به این فیلدها و متدهای public توی کلاس‌های دیگه باعث میشه که به راحتی بشه مقدار اون‌ها رو عوض کرد.

اینجاست که encapsulation ظاهر می شه. این اصل، یه خاصیت داره و اونم اینه که وضعیت داخلی آبجکت‌ها، مثلاً فیلدهاشون رو، hide می کنه. به این خاصیت encapsulation، اصطلاحاً data hiding می گن. نترسین! توی قالب کد همه رو می بینیم.

چطوری توی کدمون Encapsulation رو به کار ببریم؟

تا الان، شماها می‌دونین قالب کلی یک کلاس و نحوه‌ی تعریف کردن و ساختنش چطوریه. حالا، می‌خوایم ببینیم چطوری data hiding رو می‌تونیم اعمال کنیم.

کلیدواژه‌ی private

فرض کنین می‌خوایم یه باغ وحش رو شبیه سازی کنیم. یه کلاس به اسم Zoo تعریف می‌کنیم و این فیلدها رو توش تعریف می‌کنیم (لطفاً از این‌جا به بعد همه‌ی کدها رو داخل IntelliJ ران کنین):

```
public class Zoo {
    private String name;
    private ArrayList<Animal> animals;

    public Zoo() {
    }
}
```

حالا یه کلاس Main تعریف کنین و یه آبجکت از کلاس Zoo بسازین و به صورت زیر فیلد name رو بهش مقدار بدین:

```
public class Main {
    public static void main(String[] args) {
        Zoo zoo = new Zoo();
        zoo.name = "Woodland Park Zoo" ;
    }
}
```

به ارور خوردین! تازه جالب‌تر هم می‌شه. وقتی خودتون بخواین بعد اسم آبجکت zoo، dot notation، بزنین، intellisense اصلاً حتی این فیلد رو بهتون پیشنهاد نمی‌ده؛ یه دور امتحان کنین!

یه سوال، به نظرتون توی وراثت، وقتی کلاس پدر یه فیلد private داره، داستان چطوریه؟

فرقی نداره! فیلدهای private توی یک کلاس، حتی داخل کلاس‌های فرزندش هم به صورت مستقیم در دسترس نیستن و نمی‌شه صداشون زد!

خب تکلیف متدهای private چیه؟ اونا هم دقیقاً قضیه‌شون مشابه فیلدهاست. متدهای private یک کلاس داخل یک کلاس دیگه، حتی داخل کلاس‌های فرزندش، قابل دسترسی نیستن! بچه‌ها حواستون باشه که این متدها و فیلدهای private، داخل کلاس خودشون قابل دسترسی هستن.

حالا، می‌دونیم که یه دسته‌ی خاص از متدها توی هر کلاسی، constructorها هستن. به نظرتون می‌شه این متدها رو هم private کرد؟

جواب آره‌ست؛ خصوصاً وقتی بخواین آبجکت ساختن از کلاستون رو محدود کنین. به طبع نمی‌تونین با این constructor آبجکت بسازین. یه مثال خیلی خوبش، کلاس Math خود جاوا ست که constructorاش private هستش و آبجکتی ازش ساخته نمی‌شه و همه‌ی متدهاش هم static هستن! داخل IntelliJ این کلاس رو import کنین و بعد روی Math کلیک راست کنین و از طریق go to، گزینه‌ی Declaration or usages رو انتخاب کنین تا به سورس کلاس برسین؛ حالا به این توجه کنین:

```

126
127 public final class Math {
128
129     // Don't let anyone instantiate this class.
132     private Math() {}
133
134     // Other to e, the base of the natural logarithms.
138     public static final double E = 2.718281828459045;
139
140     // The double value that is closer than any other to pi (π), the ratio of the circumference of a circle to
141     // its diameter.
145     public static final double PI = 3.141592653589793;
146
147     // The double value that is closer than any other to tau (τ), the ratio of the circumference of a circle to
148     // its radius.
149     // API Note: The value of pi is one half that of tau; in other words, tau is double pi .
150     // Since: 19
158     public static final double TAU = 2.0 * PI;
159
160     // Constant by which to multiply an angular value in degrees to obtain an angular value in radians.
164     private static final double DEGREES_TO_RADIAN = 0.017453292519943295;
165
166     // Constant by which to multiply an angular value in radians to obtain an angular value in degrees.
170     private static final double RADIANS_TO_DEGREES = 57.29577951308232;

```

توی این‌جا یه چیز جالب دیگه هم داریم؛ **final class**! قبلاً با کلیدواژه‌ی final آشنا شدین. حالا وقتی این keyword رو برای یک کلاس میاریم، به این معنی هستش که اون کلاس دیگه قابل ارث‌بری نیستش!

راستی اگر هم روزی روزگاری بخواین از کلاسی که constructor اش private هست، آبجکت بسازین، اولاً باید حتماً داخل همون کلاس آبجکت بسازین؛ دوماً نیاز دارین که یا داخل یه متد دیگه ای این کار رو انجام بدین یا از تابع () getInstance استفاده کنین. برای توضیحات بیش تر این مورد می تونین [این ویدیوی کوتاه](#) رو ببینین.

حالا یه سوالی که مطرح می شه اینه که آیا کلاس ها هم می تونن private باشن؟

به طور کلی، آره. اما نه هر کلاسی! فقط وقتی دارین nested class تعریف می کنین، فقط inner class ها تون می تونن private باشن و **دسترسی بهش هم فقط داخل outer class ممکنه**. همچنین امکان ارث بری ازشون هم وجود نداره؛ **عین final class!** درضمن outer class ها تون نمی تونن private باشن. زمانی هم که مثلاً ۳ تا nested class دارین (که به طبع بیرونی ترینشون می شه outer class)، هم می تونین هردو تا inner class رو private کنین هم یکیشون رو. زمانی هم که کلاس عادی دارین، کلا private کردنش بی معنیه، چون علناً دارین استفاده از اون کلاس رو غیر ممکن می کنین. خب، تبریک می گم، شما تونستین data hiding انجام بدین. اما سوالی که مطرح می شه اینه که چطوری به این. متدها و فیلدهای private دسترسی داشته باشیم؟ هربار بریم دوباره اون فیلد، متد و ... رو public کنیم و اصطلاحاً صورت مسئله رو پاک کنیم؟ قطعاً نه!

برای دسترسی و تغییر **فیلدهای private** به ترتیب از متدهای getter و setter استفاده می کنیم.

getter

خیلی خلاصه، متدهای getter مقدار فیلد private رو برمی گردونن؛ درواقع اسمشون هم همین معنی رو میده (getter). موقع پیاده سازی هم برای اینکه معلوم باشه یک متد، getter هستش، اسمش رو با get شروع میکنیم. خب، وقتشه بریم سراغ کد.

لطفاً اول کلاس Animal رو به این صورت تعریف کنین و متدهای getter واسش قرار بدین:

```
import java.util.ArrayList;
public class Animal {
    private String name;
    private int age;
    private String species;

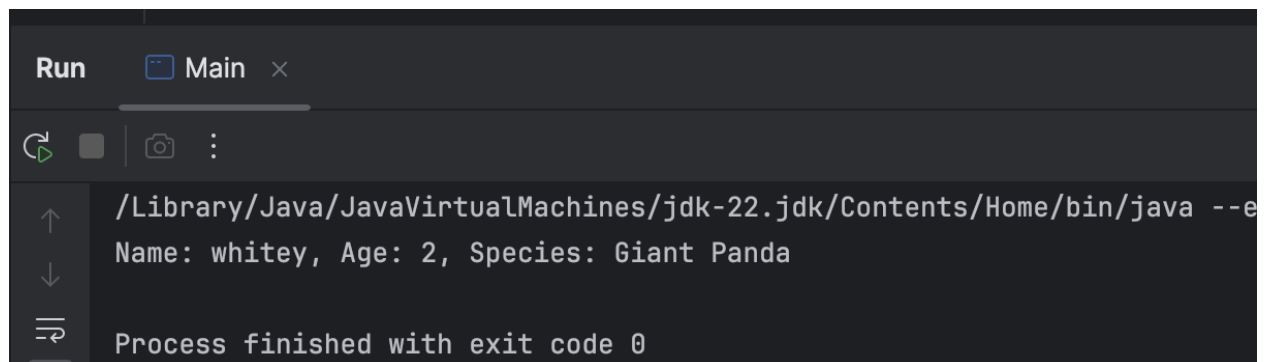
    public Animal(String name, int age, String species) {
        this.name = name;
        this.age = age;
    }
}
```

```

        this.species = species;
    }
    public String getName() {
        return name;
    }
    public String getSpecies() {
        return species;
    }
    public int getAge() {
        return age;
    }
    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Species: " + species;
    }
}

```

یه سوال، چرا متد `toString()` رو `override` کردیم؟ توی داکيومنت هفته‌ی پیش دیدین که وقتی با متد `toString()` کلاس `Object`، آبجکت یه کلاسی که نوشتین رو چاپ می‌کنین، خروجیتون چطوره. حالا برای اینکه بتونین اطلاعات قابل خوندن و مورد نیاز از آبجکتتون رو به شکل درست چاپ کنین، کافیه این متد رو `override` کنین و اونطوری که می‌خواین، خروجی رو برگردونین. مثلاً، داخل کلاس `Main` یه آبجکت از این کلاس بسازین و `toString()` رو واسش کال کنین و پرینت کنین؛ خروجی همچین چیزیه:



```

Run    Main x
Name: whitey, Age: 2, Species: Giant Panda
Process finished with exit code 0

```

کلاس `Panda` هم به عنوان فرزندش تعریف کنین و متدهای `getter` رو بذارین:

```

public class Panda extends Animal {
    private String whatTheyEat;
    private int numberOfChildren;

    public Panda(String name, int age, String species, String whatTheyEat,
int numberOfChildren) {
        super(name, age, species);
        this.whatTheyEat = whatTheyEat;
        this.numberOfChildren = numberOfChildren;
    }
}

```



```

    }
    public int getNumberOfChildren() {
        return numberOfChildren;
    }
    public String getWhatTheyEat() {
        return whatTheyEat;
    }
}

```

حالا، داخل کلاس Main کد زیر رو بنویسین و ران کنین:

```

public static void main(String[] args) {
    Panda panda = new Panda("whitey", 2, "Giant Panda", "Bambo", 0) ;
    System.out.println(panda.getName());
    System.out.println(panda.getAge());
    System.out.println(panda.getSpecies());
    System.out.println(panda.getWhatTheyEat());
    System.out.println(panda.getNumberOfChildren());
}

```

خب، همه ی اطلاعات واستون چاپ شد!

تمام متدهایی که توی کلاس Animal و Panda با get شروع می شن، getterهامون هستن که به درستی دارن مقدار فیلدهای آبجکت panda رو برمی گردونن. یه نکته ی ریزی هم این وسط بود، اونم این که آبجکت panda تونست از getterهای کلاس والدش، مثلاً getName() استفاده کنه؛ پس **getterها هم مثل متدهای عادی ارث بری می شن**، راحت تر بگیم، **فرقی با متد عادی ندارن**، صرفاً وظیفه شون برگردوندن مقدار این فیلدهاست.

setter

وظیفه ی این متدها، مقداردهی و آپدیت کردن/تغییر دادن مقدار فیلدهای private هستش. نکته ی مهم پشت منطق استفاده از setter اینه که شما سطح دسترسی فیلدتون رو محدود کردین تا مقدارش رو هرکس و هرجایی نتونه عوض کنه. پس منطقاً باید داخل این دسته از متدها، یه شرطی برای بررسی valid بودن اون مقداری که می خواین ست کنین، قرار بدین. موقع پیاده سازی هم برای اینکه معلوم باشه یک متد، setter هستش، اسمش رو با set شروع می کنیم.

حالا به کلاس Animal این متدهای setter رو اضافه کنین:

```

public void setSpecies(String species) {
    this.species = species;
}
public void setAge(int age) {

```

```

    if (age >= 0)
        this.age = age;
    else{
        System.out.println("Age can't be negative");
    }
}

public void setName(String name) {
    if (Character.isUpperCase(name.charAt(0)))
        this.name = name;
    else{
        System.out.println("Name should start with an upper-case letter");
    }
}
}

```

و کلاس Panda رو هم به این صورت تغییر بدین و متدهای setter رو اضافه کنین:

```

public void setNumberOfChildren(int numberOfChildren) {
    if (numberOfChildren >= 0)
        this.numberOfChildren = numberOfChildren;
    else{
        System.out.println("Number of children can't be negative");
    }
}

public void setWhatTheyEat(String whatTheyEat) {
    this.whatTheyEat = whatTheyEat;
}
}

```

و این قسمت رو به کلاس Main اضافه کنین و نتیجه رو ببینین.

```

System.out.println("-----");
panda.setName("cutie");
panda.setAge(1);
panda.setSpecies("Red Panda");
panda.setWhatTheyEat("mushrooms");
panda.setNumberOfChildren(1);
System.out.println(panda.getName());
System.out.println(panda.getAge());
System.out.println(panda.getSpecies());
System.out.println(panda.getWhatTheyEat());
System.out.println(panda.getNumberOfChildren());

```

کلیدواژه‌ی protected

این keyword به این صورته که اگر مثلاً فیلدی رو protected تعریف کنیم، فقط کلاس‌های فرزندش و همون پکیج بهش دسترسی دارن، ولی بقیه‌ی کلاس‌ها، دسترسی ندارن و باید از getter برای دسترسی و setter برای تغییر دادنش استفاده کنن. به عبارتی، یه چیزی بین public و private ته.

شما فهمیدین وقتی یه فیلد یا متد private باشه، حتی بچه‌هاش هم به اون دسترسی نخواهند داشت. برای اینکه این مشکل حل بشه، از protected استفاده می‌کنیم.

حالا کلاس Animal رو به این صورت تعریفش کنین:

```
import java.util.ArrayList;
public class Animal {
    protected String name;
    protected int age;
    protected String species;

    public Animal(String name, int age, String species) {
        this.name = name;
        this.age = age;
        this.species = species;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Species: " + species;
    }
}
```

کلاس Panda به این شکل تغییرش بدین:

```
public class Panda extends Animal {
    protected String whatTheyEat;
    protected int numberOfChildren;
    public Panda(String name, int age, String species, String whatTheyEat,
int numberOfChildren) {
        super(name, age, species);
        this.whatTheyEat = whatTheyEat;
        this.numberOfChildren = numberOfChildren;
    }
    public void show(){
        System.out.println(super.name);
        System.out.println(super.age);
        System.out.println(super.species);
        System.out.println(this.whatTheyEat);
        System.out.println(this.numberOfChildren);
    }
}
```

خب، حالا بریم توی Main.

```
import java.util.ArrayList;
import java.lang.*;
public class Main {
    public static void main(String[] args) {
        Panda panda = new Panda("whitey", 2, "Giant Panda", "Bambo", 0) ;
    }
}
```

```
panda.show();
}
```

می بینیم که مشکلی پیش نیومد و خروجی گرفتیم، چرا؟ گفتیم که فیلدهای protected در کلاس فرزند قابل دسترسی هستن و پوینت ما هم اینجا نشون دادن این قضیه بود! برای اینکه تفاوتش با private رو با قشنگ ببینیم، یه دور توی کلاس Animal همه ی فیلدها رو private کنین و بعد ران بگیریم.

خب، متدهای protected چی؟ اونا هم عین فیلدها هستن؛ و فرزندهای یک کلاس و کلاس های داخل یک پکیج، بهش دسترسی دارن.

خب، به نظرتون constructorهای protected قراره چطوری رفتار کنن؟ رفتارشون کاملاً مشابه متدهای protected نه. یه دور روی همین کد امتحان کنین تا ببینیم!

سوالی که ایجاد می شه اینه که تکلیف کلاس های protected چیه؟ خب، یه چیزی شبیه کلاس private نه! شما فقط زمانی می تونین کلاس protected داشته باشین که nested class دارین و فقط می تونین inner class رو protected تعریف کنین. این کلاس ها قابل ارث بری **هستن**! همچنین، از طریق کلاسی که داخل همون پکیج باشه و یا کلاس های فرزندش، می تونین به این inner class های protected دسترسی داشته باشین.

package-private

وقتی شما صراحتاً مشخص نکنین که فیلدها، متدها یا کلاس هاتون public، private یا protected هستن، اصطلاحاً میگن default access هستن؛ و package-private ها در واقع default access modifier هستن. این به معنیه که این فیلد، کلاس و متدها فقط توی همون پکیج قابل دسترسی ان. بریم سر کدش.

یه پکیج به اسم zoo بسازین و به ترتیب، کلاس های Zoo، Animal، Panda و Main رو به این صورت تعریف کنین:

```
package zoo;
import java.util.ArrayList;
public class Zoo {
    String name;
    ArrayList<Animal> animals;
```

```
public Zoo() {  
    }  
}
```

```
package zoo;  
import java.util.ArrayList;  
public class Animal {  
    String name;  
    int age;  
    String species;  
  
    public Animal(String name, int age, String species) {  
        this.name = name;  
        this.age = age;  
        this.species = species;  
    }  
    @Override  
    public String toString() {  
        return "Name: " + name + ", Age: " + age + ", Species: " + species;  
    }  
}
```

```
package zoo;  
public class Panda extends Animal {  
    String whatTheyEat;  
    int numberOfChildren;  
  
    public Panda(String name, int age, String species, String whatTheyEat,  
int number_of_children) {  
        super(name, age, species);  
        this.whatTheyEat = whatTheyEat;  
        this.numberOfChildren = numberOfChildren;  
    }  
}
```

```
import zoo.Panda;  
import zoo.Zoo;  
import java.util.ArrayList;  
import java.lang.*;  
public class Main {  
    public static void main(String[] args) {  
        Panda panda = new Panda("whitey", 2, "Giant Panda", "Bambo", 0) ;  
        System.out.println(panda.name);  
        System.out.println(panda.age);  
        System.out.println(panda.species);  
    }  
}
```

```
System.out.println(panda.whatTheyEat);  
System.out.println(panda.numberOfChildren);  
}  
}
```

خب با مشکل قاعدتا مواجه شدین. علتش اینه که کلاس Main داخل پکیج zoo نیستش. حالا به دور کلاس Main رو هم به پکیج zoo اضافه کنین نتیجه رو ببینین. این دفعه خروجی گرفتن و خب، این دقیقا مفهوم package-private هستش!

رفتار متدها و کلاسها هم توی package-privateها، مشابه فیلدهاست.

چه چیزی یاد گرفتیم؟

با خوندن این داکيومنت، فهميديم:

- encapsulation چي هست و چرا استفاده مي‌شه.
- كليدواژه‌هاي private و protected و public و مفهوم package-private چه فرقي دارن.
- متدهاي getter و setter چي هستن.