به نام خدا

آموزش سی و ششم

اهداف آموزشی این قسمت عبارتند از:

۱. معرفی Modifier ها در زبان برنامه نویسی جاوا
۲. مقایسه سطوح دسترسی با یکدیگر

در این قسمت قصد داریم تا به بررسی Access Control یا "مدیریت سطح دسترسی" به کلاس ها، متغیر ها و متد ها بپردازیم. در واقع این مبحث زمانی اهمیت دوچندانی پیدا می کند که تیمی از برنامه نویسان بخواهند به صورت گروهی روی پروژه ای کار کنند.

فرض کنیم که قرار است به عنوان سرپرست گروهی از برنامه نویسانی باشیم که روی طراحی یک نرم افزار کار می کنند. وظیفه ای که مدیر پروژه بر عهده ما گذاشته است این می باشد که کنترل کنیم که هر برنامه نویسی بنا به وظیفه ای که دارد تا چه حدی به دیگر کلاس ها، متغیر ها و متد هایی که توسط دیگر برنامه نویسان تولید می شود می تواند دسترسی داشته باشد. حال اگر ما به عنوان سرپرست مابقی برنامه نویسان کار خود را به خوبی انجام دهیم، به نظر می رسد که مشکلی حادی در این رابطه بوجود نیاید. حال فرض کنیم که ما آن طور که باید و شاید در مورد Access Control در زبان برنامه نویسی جاوا آگاهی نداریم. در این جا است که پروژه ما با مشکلات جدی مواجه خواهد شد چرا به طور مثال برنامه نویسی که نمی بایست به یک متغیر از جنس int دسترسی داشته باشد حال این امکان برایش فراهم شده و ایشان هم بدون هیچ سوء قصدی مقدار اولیه آن متغیر را تغییر داده اند و نتیجه ای که برنامه در حین کار می بایست داشته باشد به نتیجه ای غیر قابل باور تبدیل شده است.

برای درک بهتر سطح دسترسی در زبان برنامه نویسی جاوا پروژه ای تحت عنوان 36^{th} Session به معنی **"جلسه سی و ششم"** ایجاد می کنیم.

در حقیقت با قرار دادن کلید واژه public پیش از یک کلاس، متد و یا متغیر این دستور را به برنامه خود می دهیم که این کلاس، متد و یا متغیر در دیگر بخش های برنامه و توسط دیگر کلاس ها

قابل دسترسی است. به عبارت دیگر از هر جای دیگر برنامه من جمله بخش های دیگر همان کلاس، دیگر پکیج ها، کلاس های ارث بری شده از آن می توان به عناصر public دسترسی داشته و در صورت لزوم تغییری در آن ها ایجاد کرد.

کلید واژه protected حاکی از آن است که عنصری که این کلید واژه قبل از آن قرار گرفته است در همان کلاس، دیگر پکیج ها و کلاس های ارث بری شده از آن قابل دسترسی خواهد بود.

کلید واژه private این معنا را می رساند که مثلا متغیری که private است فقط و فقط در همان کلاس قابل دسترسی خواهد بود.

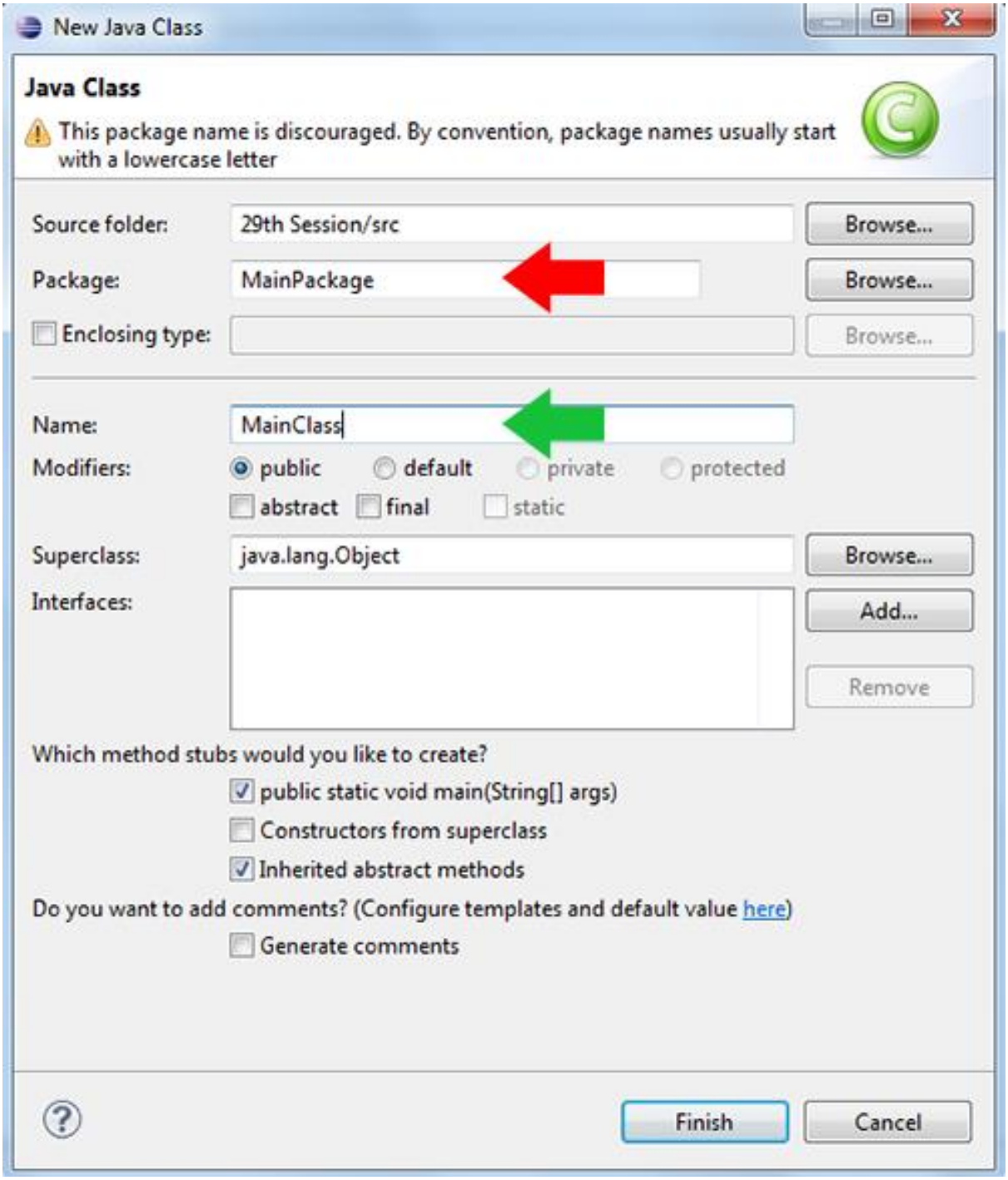
حال زمانیکه هیچ کلید واژه محدود کننده ای قبل از نام یک کلاس، متد و یا متغیر نباشد، این سطح دسترسی Package Access نامیده می شود و بدان معنا است که عنصری که فاقد هرگونه public, protected, private باشد در همان کلاس و یا دیگر کلاس های موجود در همان پکیج قابل دسترسی خواهد بود.

برای روشن شدن تفاوت مابین این چهار سطح دسترسی می توان از جدول زیر کمک گرفت:

	Class	Package	Subclass	Others
public	√	√	√	√
protected	√	√	√	—
	√	√	—	—
private	√	—	—	—

حال برای درک بهتر کلیه سطوح دسترسی نیاز داریم تا یک پروژه کوچک تعریف کرده و بر اساس آن یک برنامه بنویسیم. فرض کنیم که می خواهیم دو کلاس داخل یک پکیج ایجاد کنیم که یکی از آن ها دارای محدودیت بیشتری نسبت به کلاس دیگری است.

در پروژه ای که در ابتدای این آموزش ایجاد کردیم یک کلاس تحت عنوان MainClass به معنی کلاس اصلی ایجاد می کنیم. پیش از زدن دکمه Finish می بایست یک نام برای پکیج خود نیز انتخاب کنیم. برای همین منظور نام MainPackage را نیز به معنی **"پکیج اصلی"** در نظر می گیریم. برای روش تر شدن مطلب به تصویر زیر توجه کنید:



همانطور که در این تصویر مشاهده می شود جایی که با فلش قرمز مشخص شده می بایست برای نام پکیج نوشته شود و جایی که با فلش سبز رنگ مشخص شده می بایست برای نام کلاس پر شود.

از آنجا که این کلاس قرار است کلاسی باشد که برنامه در آن شروع شود از این رو در حین ساختن آن می بایست گزینه public static void main(String[] args) را تیک بزنیم.

در این مرحله کد ما می بایست به شکل زیر باشد:

```
package MainPackage;

public class MainClass {
	public static void main (String[] args){

	}

}
```

همانطور که در کد بالا ملاحظه می شود در خط اول نام پکیجی که وارد کردیم وارد برنامه شده است. حال اقدام به ایجاد یک کلاس دیگر تحت عنوان SecondClass به معنی **"کلاس دوم"** می کنیم. از آنجا که این کلاس قرار نیست که به منزله نقطه شروع برنامه باشد گزینه public static void main(String[] args) را تیک نمی زنیم. کد مربوط به کلاس دوم ما می بایست به شکل زیر باشد:

```
package MainPackage;

public class SecondClass {

}
```

حال می خواهیم تا کلاس اول ما یکسری از خصوصیات کلاس SecondClass را به ارث ببرد از این رو کد MainClass را به شکل زیر بازنویسی می کنیم:

```
package MainPackage;

public class MainClass extends SecondClass{
public static void main (String[] args){

	}

}
```

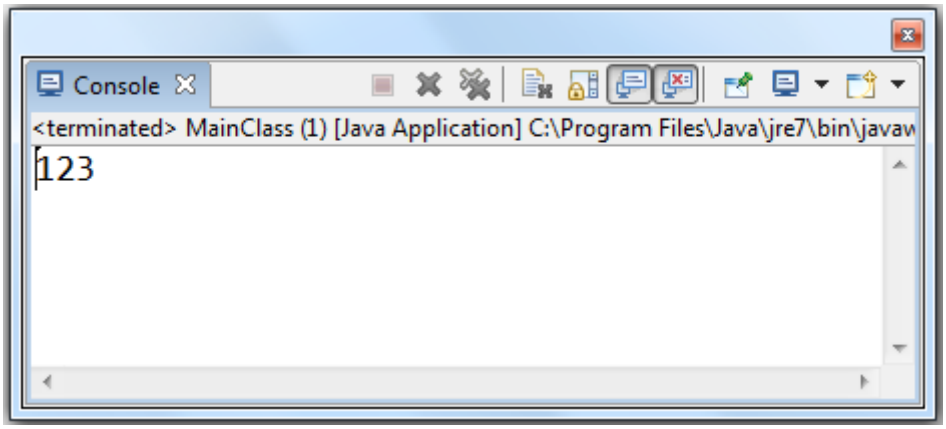
حال کلاس دوم خود را به شکل زیر تکمیل می کنیم:

```
package MainPackage;

public class SecondClass {
	int number = 123;
	public void testMethod() {
		System.out.println(number);
	}
}
```

در این کد ما یک متغیر از جنس int تحت عنوان number به معنی **"شماره"** ایجاد کرده سپس یک متد به اسم testMethod ایجاد کرده و در آن دستور داده ایم که مقدار متغیر ما تحت عنوان number در پنجره Console نمایش داده شود. البته این کلاس از آن جا که نقطه آغازین یا همان متد main را ندارد چیزی اجرا نخواهد کرد. حال در کلاس MainClass یک شیئ از روی این کلاس ایجاد می کنیم و آن را در متد main اجرا می کنیم. کد ما در کلاس MainClass به شکل زیر خواهد بود:

```
package MainPackage;

public class MainClass extends SecondClass {
	public static void main(String[] args) {
		SecondClass myObject = new SecondClass();
		myObject.testMethod();
	}
}
```

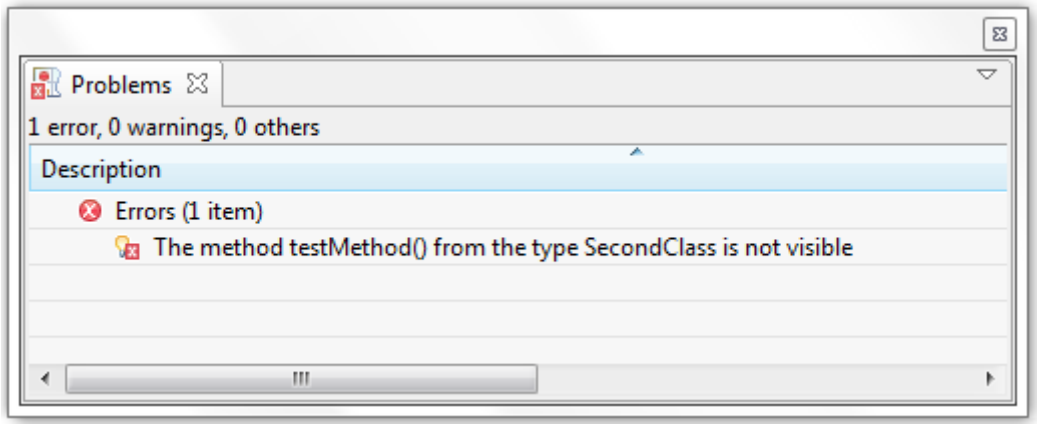
همانطور که در آموزش های پیشین توضیح داده شد، از روی کلاس SecondClass یک شیئ تحت عنوان myObject به معنی **"شیئ من"** ساخته سپس در خط بعدی، متد قرار گرفته در آن کلاس را فرا می خوانیم. حال اگر برنامه را اجرا کنیم، تصویر زیر مشاهده خواهد شد:



در واقع عدد ۱۲۳ مقدار متغیری تحت عنوان number است که در کلاس SecondClass ایجاد شده است. حال می خواهیم تا از قصد سطح دسترسی متد ایجاد شده در SecondClass را محدود کنیم. به عبارت دیگر سطح دسترسی آن را از نوع public به نوع private تغییر می دهیم. کد ما به شکل زیر بازنویسی خواهد شد:

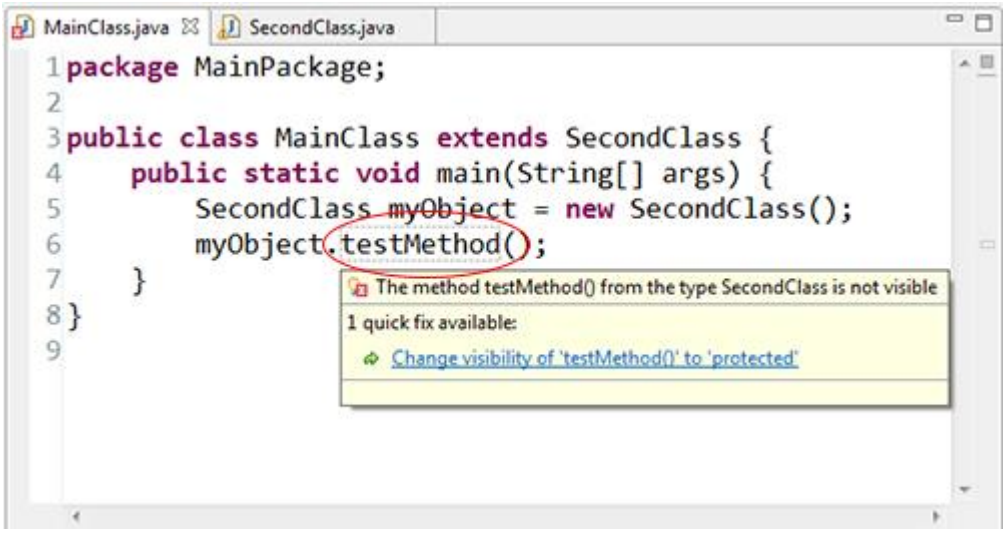
```
package MainPackage;

public class SecondClass {
	int number = 123;
	private void testMethod() {
		System.out.println(number);
	}
}
```

حال به محض اینکه فایل خود را Save کنیم در پنجره Problems خطای زیر ایجاد خواهد شد:



این Error حاکی از آن است که متدی تحت عنوان testMethod از کلاس SecondClass قابل روئیت نمی باشد(چنانچه پنجره Problems باز نباشد می توان از منوی اصلی محیط برنامه نویسی اکلیپس گزینه Window را انتخاب کرده سپس گزینه Show View و در نهایت گزینه Problems را کلیک نماییم).

حال اگر به MainClass نگاهی بیندازیم می بینیم که آنجا هم این مشکل خود را نشان خواهد داد:



در واقع به محض Save کردن فایل خود دور نام متد testMethod یک نقطه چین قرار خواهد گرفت و با قرار دادن نشانگر موس خود روی آن می بینیم که اکلیپس به ما پیشنهاد می دهد که سطح دسترسی مرتبط با این متد را تغییر دهیم تا مشکل برنامه رفع گردد.

در این پروژه دیدیم که به چه سهولتی می توانیم سطح دسترسی به متدها را در برنامه خود تغییر دهیم.

پس از مطالعه این آموزش انتظار می رود بتوانیم به سؤالات زیر پاسخ بدهیم:

۱. تفاوت سطوح دسترسی public, private, protected چیست؟
۲. اگر برای یک کلاس، متد و یا متغیر هیچ گونه سطح دسترسی تعریف نکنیم این به چه معنا است؟

در آموزش قسمت بعد، خواهیم دید که متدها در زبان برنامه نویسی جاوا از چه اجزایی تشکیل شده اند و چگونه می تواند یک متد را از پایه ساخت.