

به نام خدا

آموزش چهارم

پس از آشنایی با مفاهیم شئی گرایی در برنامه نویسی، دو مسئله ای را که می تواند برای برنامه نویسان مبتدی مفید واقع شود را مطرح خواهیم کرد. این دو مسئله عبارتند از Error هایی که ما در برنامه نویسی با آن ها مواجه خواهیم شد و همچنین نحوه Comment کردن یا مخفی کردن بخشی از اطلاعات.

در مقدمه این سری از آموزش های زبان برنامه نویسی جاوا، توضیح مختصری از Error ها یا مشکلاتی که برنامه نویسان مکرراً با آن ها مواجه می شوند آورده شد. ابتدا به مرور مطالب گذشته پرداخته و سپس موارد تکمیلی را توضیح خواهیم داد.

مشکلات یا Error ها در زبان برنامه نویسی جاوا به چند دسته تقسیم می شوند: گروه اول Error هایی هستند که از نوع Compile-time Error می باشند. به طور مثال دستور `system.out.println();` در مقایسه با `System.out.println();` اشتباه است چرا که حرف اول واژه `system` به صورت کوچک نوشته شده است و همین مسئله موجب می گردد که برنامه Compile نشود. خبر امیدوار کننده اینجا است که این گروه از مشکلات توسط خود نرم افزاری که برنامه خود را با آن می نویسیم تشخیص داده شده و به ما اخطار داده می شوند و به سادگی می توان آن را رفع کرد. برای روشن شدن این مطلب به ذکر چند مثال تصویری به همراه توضیحات خواهیم پرداخت.

همانطور که در آموزش های قبلی فرا گرفتیم، پروژه ای در محیط برنامه نویسی جاوا ایجاد کرده و یک Class با نام `CompileTimeErrors` یا هر نام دیگری در آن ایجاد می کنیم (فقط به خاطر داشته باشیم که نام Class ما با نام فایل `java`. که می سازیم

دقیقاً یکی باشد که در غیر این صورت برنامه ما با مشکل مواجه خواهد شد). حال می بایست کدی مشابه کد زیر داشته باشیم:

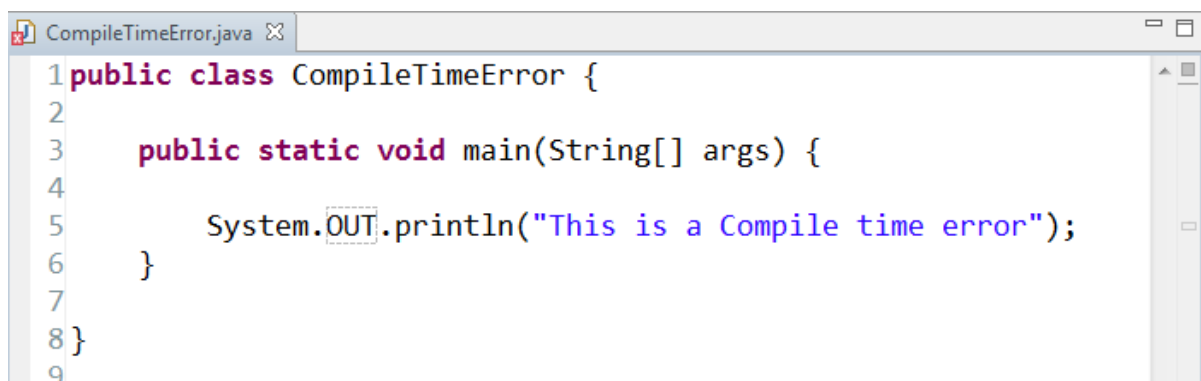
```
public class CompileTimeErrors {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
    }  
}
```

حال می خواهیم جمله This is a Compile time error را روی صفحه مانیتور نمایش دهیم پس می بایست کد خود را به شکل زیر ویرایش نماییم (لازم به ذکر است که Comment های کد را که هیچ تاثیری در نحوه ی اجرای برنامه ما ندارند را حذف می کنیم. در بخش بعدی پیرامون انواع Comment ها به طور مفصل توضیح خواهیم داد):

```
public class CompileTimeErrors {  
  
    public static void main(String[] args) {  
        System.out.println("This is a Compile time error");  
    }  
}
```

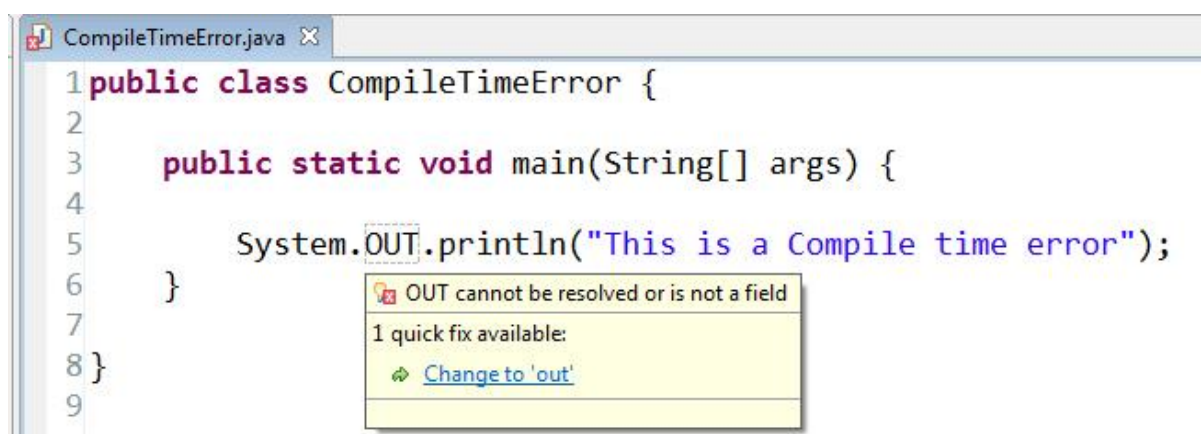
در واقع کد فوق یک برنامه کامل و صحیح جاوا است که بدون هیچ مشکلی اجرا خواهد شد. در این جا ما قصد داریم تا عمداً تغییری در کد ایجاد کنیم تا Compiler از ما ایراد بگیرد. به همین منظور واژه out که می بایست حتماً با حروف کوچک نوشته شود

را به صورت OUT می نویسیم. به محض اینکه ما واژه ی out را به OUT تغییر دهیم، با تصویر زیر مواجه خواهیم شد:



```
1 public class CompileTimeError {
2
3     public static void main(String[] args) {
4
5         System.OUT.println("This is a Compile time error");
6     }
7
8 }
9
```

ما در تصویر فوق از روی دو مورد می توانیم متوجه شویم که برنامه ما دارای Error است. محیط برنامه نویسی اکلیپس دور واژه ی مشکل زا که در اینجا OUT است یک نقطه چین قرار داده است. از سوی دیگر در کنار نام فایل java. دو علامت ستاره مشاهده می شود که نشانگر ایرادی در برنامه است. حال با قرار دادن نشانگر موس خود روی واژه OUT اکلیپس پیامی را به ما نشان می دهد که به شکل زیر است:



```
1 public class CompileTimeError {
2
3     public static void main(String[] args) {
4
5         System.OUT.println("This is a Compile time error");
6     }
7
8 }
9
```

OUT cannot be resolved or is not a field
1 quick fix available:
➡ Change to 'out'

این یکی از بخش های لذت بخش کار کردن با نرم افزار اکلیپس است. این محیط برنامه نویسی قدرتمند مشکل را پیدا کرده و در جهت رفع آن تعدادی پیشنهاد می دهد(در اینجا صرفاً یک پیشنهاد بیشتر به ما نمی دهد). چنانچه به خط آبی رنگ توجه کنیم، می بینیم که به ما پیشنهاد می شود که واژه OUT را به out تغییر دهیم که با این کار مشکل ما رفع خواهد شد. لازم به ذکر است که مثال فوق به منزله یکی از ساده ترین انواع Error ها در محیط برنامه نویسی جاوا است. در واقع یکسری از Error ها هستند که تا حدودی پیچیده تر هستند و اکلیپس برای رفع مشکل، بیش از یک مورد به ما پیشنهاد می دهد که در چنین حالتی کار ما به عنوان برنامه نویس این است که تک تک پیشنهادات را مورد بررسی قرار داده و سپس تغییر مورد نیاز را اعمال کنیم.

نوع دیگر از Error ها تحت عنوان Unchecked Runtime Exception شناخته می شوند. در این صورت ما هیچ گونه Compile-time Error نداریم اما برنامه هم به طور کامل اجرا نمی شود. در واقع برنامه ای که ما نوشته ایم کاری را از جاوا می خواهد انجام دهد که جاوا از انجام آن ناتوان است. به طور مثال فرض کنیم که ما یک متغیر از جنس int داریم که Value اختصاص داده شده به آن 1,000,000 است. در حین نوشتن برنامه ما با هیچ گونه مشکلی مواجه نخواهیم شد اما به محض اینکه اقدام به Compile کردن برنامه می کنیم Error دریافت می کنیم. در مثال فوق اشکال کار در اینجا است که ما مابین ارقام یک کما قرار داده ایم از این رو جاوا قادر به Compile کردن برنامه نخواهد بود. به منظور رفع این مشکل می بایست عدد فوق را به 1000000 تبدیل کنیم. در حقیقت نرم افزار اکلیپس از درک اینکه این مسئله می تواند در حین اجرا مشکل زا باشد عاجز است. برای روشن تر شدن این مسئله مثالی از دنیای واقعی می زنیم. فرض کنیم که شما از شخصی آدرس می پرسید. آن شخص به شما می گوید که مستقیم بروید تا به یک چهار راه برسید سپس به سمت راست بروید و رو به روی شما

دوره آموزش جاوا

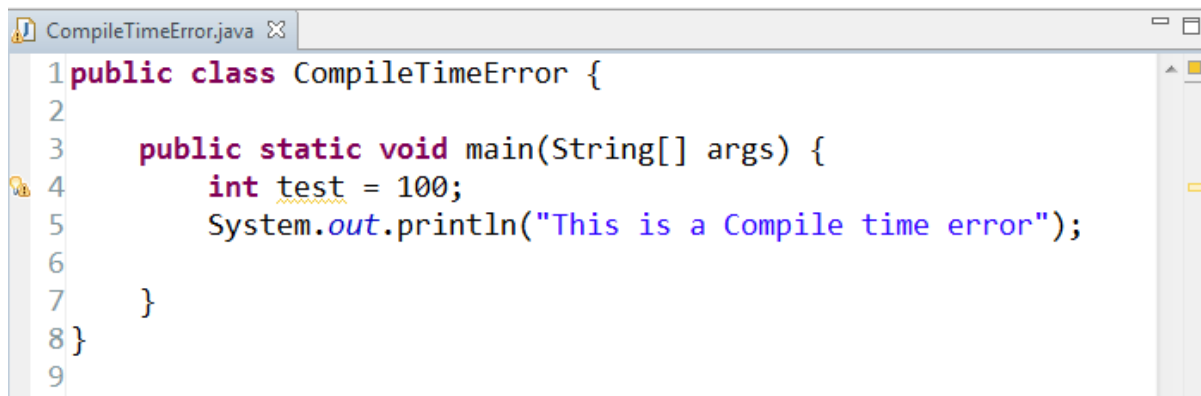
کلیه حقوق متعلق به وب سایت نردبان است.

مدرس: بهزاد مرادی

یک سینما قرار دارد. حال شما همانطور که آن شخص گفت مسیر را طی می کنید. مستقیم می روید. به چهار راه که رسیدید به سمت راست می روید. به رو به رو که نگاه می کنید به جای یک سینما یک دیوار سیمانی رو به روی شما قرار دارد. به عبارت دیگر شما از اول نمی دانستید که آدرسی که به شما داده شده اشتباه است. زمانیکه اینگونه Error ها اتفاق می افتند نیز همین طور است. از بدو امر جاوا نمی داند که در حین اجرای برنامه در وسط کار مشکلی قرار است اتفاق افتد از همین رو نمی تواند پیش بینی کند که برنامه در حین Compile با مشکل مواجه خواهد شد.

دسته دیگری از Error ها از نوع Logical یا منطقی هستند. در چنین مواقعی نرم افزار اکلپس هیچ گونه خطایی را نشان نخواهد داد و برنامه هم به طور کامل اجرا خواهد شد اما پاسخی که نرم افزار قرار بود به ما بدهد صحیح نیست. فرض کنیم در مثال فوق شخصی که به ما آدرس می دهد به جای اینکه به ما بگوید به سمت راست برویم بگوید که به سمت چپ برویم. این دسته از Error ها به منزله چالش بر انگیزترین Error ها می باشند به طوری که رفع کردن آن ها هم ممکن است با دشواری زیادی همراه باشد (شاید مجبور باشید هفته ها وقت صرف کنید تا مشکل را بیابید). نکته ای که بایستی مد نظر قرار داده شود این است که خیلی از این دسته از Error ها خیلی اوقات اصلاً مورد توجه قرار نمی گیرند و در آینده مشکلات زیادی را برای ما به بار خواهند آورد.

دسته آخر Error ها که بهتر است نام آن ها را Error نگذاریم بلکه Warning یا هشدار بنامیم تحت عنوان Compile-time Warning شناخته می شوند. به طور کلی این دسته از هشدار ها به مهمی Error ها نمی باشند. در واقع زمانی که محیط برنامه نویسی اکیپس شاهد مسئله مشکوکی باشد، یک آیکون زرد رنگ با علامت تعجب مقابل خطی که به آن مشکوک است نمایش می دهد. به مثال زیر توجه فرمایید:



```
1 public class CompileTimeError {
2
3     public static void main(String[] args) {
4         int test = 100;
5         System.out.println("This is a Compile time error");
6     }
7 }
8
9
```

در مثال فوق، ما یک متغیر از جنس int ایجاد کرده ایم که نام آن test است و یک Value به میزان ۱۰۰ برای آن در نظر گرفته شده است. در مقابل خطی که متغیر در آن قرار دارد یک علامت زرد رنگ با یک علامت تعجب قرار دارد. به عبارت دیگر این یک هشدار از طرف نرم افزار اکیپس است که می خواهد به ما بگوید که این متغیر در هیچ کجای برنامه مورد استفاده قرار نگرفته است و دیگر اینکه آیا مطمئن هستید که می خواهید این متغیر را در کد خود نگه دارید. در واقع هر موقع که ما با چنین هشدار هایی مواجه شویم می توانیم نشانگر موس خود را روی آن علامت زرد رنگ نگه داشته و پیشنهادی را که اکیپس به ما می دهد را مطالعه کنیم. به تصویر زیر توجه فرمایید:

```
CompileTimeError.java x
1 public class CompileTimeError {
2
3     public static void main(String[] args) {
4         int test = 100;
5         System.out.println("This is a Compile time error");
6     }
7 }
8
9
```

The value of the local variable test is not used

نوشته ای که در قسمت زرد رنگ دیده می شود حاکی از آن است که **Value متغیری** تحت عنوان **test** در هیچ کجای برنامه مورد استفاده قرار نگرفته است.

به طور کلی مشکلات برنامه نویسی اصطلاحاً Bug نامیده می شوند و فرایند مشکل یابی و رفع آن اصطلاحاً Debugging گفته می شوند که تسلط به Debugging خیلی می تواند ما را در برنامه نویسی کمک کند.

اضافه کردن Comment به کدهای خود در حین برنامه نویسی، به طرز چشمگیری می تواند ما را در برنامه نویسی کمک کند. در حقیقت برنامه نویسان با اینکار راه را برای ویرایش کد خود در آینده آسان می کنند. همانطور که قبلاً هم اشاره شد در زبان برنامه نویسی جاوا سه مدل Comment وجود دارد که عملکرد هر کدام از آن ها را مورد بررسی قرار خواهیم داد.

```

class FirstProgram {
    /*
     * This is a multi-line comment for you!
     */
    public static void main(String[] args) {
        String text = "In the name of God";
        System.out.println(text);
    }
}

```

نوع اول از Comment ها چند خطی می باشند. به عبارت دیگر چنانچه ما بخواهیم چند خط از کد خود را از دید Compiler پنهان سازیم در خط اول علامت `/*` و در خط آخر علامت `*/` را قرار می دهیم و هر آنچه که فی مابین قرار گیرد جزو Comment حساب خواهد شد. نکته ای که در مورد این نوع از Comment ها می بایست به خاطر داشته باشیم این است که ما به عنوان برنامه نویس جاوا نمی توانیم یک Comment چند خطی را درون یک Comment چند خطی دیگر قرار دهیم چرا که با اینکار در حین اجرای برنامه با مشکل مواجه خواهیم شد.

نوع دوم از Comment ها به Comment های پایان خط معروف هستند. در واقع چنانچه در انتهای خطی که کدی را در آن نوشته ایم بخواهیم جهت یادآوری خود چیزی یادداشت کنیم از این Comment با قرار دادن علامت `//` استفاده خواهیم کرد. مثالی از این Comment را در کد زیر مشاهده خواهید کرد:

```

class FirstProgram {

    public static void main(String[] args) { //This is a method
        String text = "In the name of God";
        System.out.println(text);
    }
}

```


آنچه در مورد این نوع از Comment ها بسیار جالب توجه است این است که چنانچه بخشی از برنامه شما دچار مشکل باشد و شما به اصطلاح بخواهید برنامه را Debug کنید می توانید از این Comment استفاده کنید به این صورت که نشانگر موس خود را روی بخشی از کد که تصور می کنید مشکل زا است قرار داده، سپس دکمه کنترل را با علامت / فشار دهید. خواهید دید که آن بخش از کد شما Comment شده و به رنگ متفاوتی در خواهد آمد. حال می توانید برنامه را Compile کنید و ببینید که آیا مشکل رفع شده است یا خیر. چنانچه مشکل رفع شده بود متوجه خواهید شد که آن بخشی از کد که Comment شده است دارای مشکل است پس نیاز است که آن را رفع نمایید. چنانچه رفع نشده بود مجدداً با انتخاب آن بخش از کد دکمه کنترل و علامت / را فشار دهید تا از حالت Comment در آید و این کار را برای مابقی بخش های برنامه خود تا زمانی می توانید انجام دهید که مشکل برنامه خود را پیدا نمایید (اگر چه نام این Comment ها تحت عنوان Comment های پایان خط معروف است، اما ما می توانیم آن ها را در ابتدای یک خط از کد نیز مورد استفاده قرار دهیم).

نوع سوم از Comment ها از نوع javadoc می باشند به این معنی که با قرار دادن علامت /** در ابتدای خطی که تمایل داریم به Comment تبدیل شود و قرار دادن علامت */ در انتهای خطی Comment ما به پایان می رسد، ما توضیحاتی را در داخل کد خود درج می کنیم که برای کسانی که می خواهند کد ما مطالعه کنند ولی دانش زبان جاوا ندارند و یا برای دیگر برنامه نویسان که می خواهند کد ما را مطالعه کنند بسیار مفید خواهد بود به این صورت که از کلیه Comment های برنامه خود که از نوع javadoc هستند خروجی HTML می گیریم و کلیه Comment ها به صورت یک صفحه وب در اختیار علاقمندان قرار می گیرد (نحوه ایجاد javadoc برای برنامه نویسان مبتدی ضروری نیست بنابراین از توضیح در این باره خودداری می شود).

دوره آموزش جاوا

کلیه حقوق متعلق به وب سایت نردبان است.

مدرس: بهزاد مرادی

```

class FirstProgram {
    /**
     * This is a javadoc comment for you!
     */
    public static void main(String[] args) {
        String text = "In the name of God";
        System.out.println(text);
    }
}

```

در این آموزش با انواع Error ها و همچنین با نحوه صحیح Comment کردن بخشی از کد خود آشنا شدیم. در آموزش بعدی با انواع متغیر های در زبان برنامه نویسی جاوا و نحوه اختصاص یک Value یا مقدار به آن ها و غیره آشنا خواهیم شد.