بسم اللّه الرحمن الرحیم

آموزشگاه تحلیل داده

تخصصی ترین مرکز برنامه نویسی و دیتابیس در ایران

عملکرد پشته در جاوا

گردآورنده : مهندس افشین رفوآ

آدرس آموزشگاه : تهران - خیابان شریعتی - بالا تر از خیابان ملک - جنب بانک صادرات - پلاک 561 - واحد 7
88146323 - 88446780 - 88146330

عملکرد پشته در جاوا

در جریان عادی یک برنامه، وقتی ماشین مجازی جاوا (Java Virtual Machine) در حال اجرای کد شما می باشد، یک متد بعد از دیگری اجرا خواهد شد که با متد main آغاز می شود. وقتی در ابتدای صف برنامه نویسی نوبت به یک متد رسیده باشد، گفته می شود که متد در راس stack می باشد. پس از اینکه تمام متد اجرا می شود، از stack گرفته شده تا متد دیگر در صف متدها جایگزین آن شود. برای توضیح اصول، کد برنامه ی خود را به شکل زیر تغییر دهید.

```
package errorhandling;

public class errorChecking {

    public static void main(String[] args) {

        System.out.println("Starting Main method");
        m1();
        System.out.println("End Main method");
    }

    static void m1() {
        System.out.println("Method One - m1");
        m2();
    }

    static void m2() {
            System.out.println("Method Two - m2");
    }
}
```

اکنون یک متد Main و دو متد دیگر در دست داریم که عبارتند: m1 و m2. وقتی که در ابتدا برنامه آغاز می شود، متد Main در بالای stack می باشد. به هرحال در داخل متد Main یک فراخوانی برای متد m1 وجود دارد. وقتی که این متد فراخوانده می شود، در بالای stack قرار می گیرد. پس از آن متد m1 متد m2 را فرا می خواند. وقتی که متد m2 فراخوانده می شود، در بالای stack قرا گرفته و m1 را به طور موقت کنر می

گذارد. پس از اتمام m2، کنترل دوباره به m1 بازمی گردد. وقتی m1 تمام می شود، در بالای stack خاموش می شود و کنترل دوباره به متد Main بازمی گردد.

برنامه ی خود را اجرا کرده و پنجره ی Output را بررسی کنید تا موارد چاپ شده را مشاهده کنید.

```
Starting Main method
Method One - m1
Method Two - m2
End Main method
```

اگر در متد m2 اشتباهی رخ دهد، JVM به جستجوی هرگونه کنترل خطا می پردازد، از جمله یک گروه try ... catch. اگر هیچگونه کنترل خطایی وجود نداشته باشد، Exception به m1 تحویل داده خواهد شد تا رویارویی آن با خطا را مشاهده کنید. در m1 هیچگونه بررسی خطایی وجود ندارد، بنابراین مجددا Exception به stack منتقل می شود، این بار به متد Main منتقل می شود. اگر متد Main با Exception هماهنگ نباشد، یک پیغام خطای عجیب در پنجره ی Output دریافت خواهید کرد. به عنوان یک مثال متد m2 را با مورد زیر تطبیق دهید.

```
static void m2( ) {
        int x = 10;
        int y = 0;
        double z = x / y;
        System.out.println( z );
        System.out.println("Method Two - m2");
        }
```

متد دوباره حاوی خطای تقسیم بر صفر می باشد. اکنون کد شما باید مشابه کد ما در زیر باشد.

```
package errorhandling;

public class errorChecking {

    public static void main(String[] args) {

        System.out.println("Starting Main method");
        m1();
        System.out.println("End Main method");
    }

    static void m1() {
        System.out.println("Method One - m1");
        m2();
    }

    static void m2() {
            int x = 10;
            int y = 0;
            double z = x / y;

            System.out.println( z );
            System.out.println("Method Two - m2");
    }
}
```

برنامه را اجرا کرده و مشاهده کنید که در پنجره ی Output چه اتفاقی می افتد.

```
Starting Main method
Method One - m1
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at errorhandling.errorChecking.m2(errorChecking.java:20)
        at errorhandling.errorChecking.m1(errorChecking.java:14)
        at errorhandling.errorChecking.main(errorChecking.java:8)
Java Result: 1
```

آنچه مشاهده می کنید stack trace نامیده می شود. سه خط آبی که زیر آنها خط کشیده شده به متد شما و جایی که یافت می شوند اشاره دارند.

```
package_name.class_name.method_name
```

مورد بالا جایی است که ابتدا خطا در آن اتفاق می افتد، در m2. جاوا در جستجوی این مورد بود تا توسط یک ArithmeticException کنترل شود که جایی است که در آن تقسیمات بر صفر گرفته می شوند. در m1، m2 و یا main هیچ بررسی خطایی وجود نداشت. بنابراین برنامه به کنترل کننده ی خطای پیش فرض خروجی می دهد.

متد m1 را به شکل زیر تغییر دهید.

```
try {
        System.out.println("Method One - m1");
        m2( );
        }
        catch (ArithmeticException err) {
        System.out.println( err.getMessage( ) );
        }
```

اکنون متد m2 را در یک بلوک try قرار داده ایم. در بخش catch از نوع Exception استفاده کرده ایم که در stack trace – ArithmeticException گزارش شد.

مجددا کد را اجرا کنید، پنجره ی Output صفحه ی زیر را نمایش خواهد داد.

```
Starting Main method
Method One - m1
/ by zero
End Main method
```

دقت داشته باشید که پیغام خطا با عنوان "/ by zero" چاپ شده است. تمام متد m2 اجرا نشد، اما در جایی که خطا اتفاق افتاد، متوقف شد. سپس کنترل به m1 بازگردانده شد. از آنجایی که یک بلوک catch برای رسیدگی به خطا وجود داشت، JVM نیازی به کنترل کننده ی پیش فرض ندید، اما پیغام را بین کروشه های catch چاپ کرد.

به هرحال خود برنامه متوقف نشد. کنترل به متد Main بازمی گردد که در آن متد m1 فراخوانده می شود. خط آخر در متد Main که "End Main method" را چاپ کرده، اجرا شد. این برنامه تاثیرات مهمی دارد. فرض کنید که شما به مقداری از m1 نیاز داشتید، زیرا تصمیم داشتید با آن در Main کاری انجام دهید. مقدار در آنجا نیست و ممکن است برنامه ی شما به شکلی که انتظار می رود رفتار نکند.

اما اگر در پنجره ی Output یک stack trace مشاهده می کنید، به یاد داشته باشید که اولین خط جایی است که مشکل در آن اتفاق افتاد. بقیه ی خطوط جایی است که Exception به stack تحویل داده می شود، که معمولا در متد main تمام می شود.

در بخش بعد به Logic Errors (خطاهای منطقی) خواهیم پرداخت.