#Custom Exceptions

As you saw if you clicked the full exception list link in a previous lesson, Java has a ton of built in exceptions that we can take advantage of. Some are automatically available, others need to be imported–but all together, they handle just about everything we need them to. But what happens when they don't?

That's when we write our own **custom exceptions**. Often, the reasons for writing our own exceptions are pretty straightforward: 1) we want our own exceptions to be more specific than the options Java has provided us, or, 2) the business logic of our application needs more clarity specific to our use cases than Java's built-in exceptions can cover. Whether we create them for these reasons or something else entirely doesn't really matter: we have to make them the same way, so let's get into it.

#Making Exceptions

Custom exceptions always start by **extending** Java's built-in Exception class or any of `Exception`'s subclasses. When we do this, we create what's called a **custom exception** or a **user-defined exception**. Our new exception will then have access to the superclasses from which it is extended, and offer us the opportunity to add in the extra information we need.

#Learn By Example

Let's walk through some custom exceptions and learn about the reasons for using them as we go.

#Example 1:

In this example, we have a `BankAccount` class with a `withdraw` method that throws an `InsufficientFundsException` if the amount being withdrawn is greater than the current `balance`. The `InsufficientFundsException` class has a constructor that takes in a double representing the amount needed to complete the transaction, and a getter method `getAmount` to access the value. In the `main` method, we create a `BankAccount` object, and call the withdraw method and `catch` the `InsufficientFundsException` if it's thrown. The `catch` block then prints the error message along with the amount needed to complete the transaction using the getter method.

Run the code, change some things, see if you understand how it works:

```
// Main.java
public class Main {
    public static void main(String[] args) {
        BankAccount ba = new BankAccount("12345", 1000);
        try {
            ba.withdraw(2000);
        } catch (InsufficientFundsException e) {
            System.out.println(
```

```
                    "Error: Insufficient funds. You need $" + e.getAmount() +
" more to complete this transaction.");
            }
        }
}
// BankAccount.java
public class BankAccount {

    private double balance;
    private String accountNumber;

    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public void withdraw(double amount) throws InsufficientFundsException {
        if (amount > balance) {
            double needs = amount - balance;
            throw new InsufficientFundsException(needs);
        }
        balance -= amount;
        System.out.println("Withdrawal of $" + amount + " successful. New bala
nce: $" + balance);
    }

}
// InsufficientFundsException.java
public class InsufficientFundsException extends Exception {

    private double amount;

    public InsufficientFundsException(double amount) {
        this.amount = amount;
    }

    public double getAmount() {
        return amount;
    }
```

#Example 2:

In this example, the `Order` class has a constructor that takes in the `quantity` and `price` of an order. If the `quantity` or `price` is less than or equal to zero, the constructor throws an instance of the `InvalidOrderException` class with an error message. In the `main` method of

the `Main` class, an instance of the `Order` class is created with an invalid quantity and price of -5 and 20 respectively. This triggers the `InvalidOrderException` to be thrown and caught by the try-catch block in the `main` method. The error message from the exception is then printed to the console. If you run this code, the output will be: "Invalid order quantity or price."

Again, run the code and play around with it a little:

```
// Main.java
public class Main {
    public static void main(String[] args) {
        try {
            Order order = new Order(-5, 20);
        } catch (InvalidOrderException e) {
            System.out.println(e.getMessage());
        }
    }
}
// InvalidOrderException.java
public class InvalidOrderException extends Exception {
    public InvalidOrderException(String message) {
        super(message);
    }
}
// Order.java
public class Order {
    private int quantity;
    private double price;

    public Order(int quantity, double price) throws InvalidOrderException {
        if (quantity <= 0 || price <= 0) {
            throw new InvalidOrderException("Invalid order quantity or price."
);
        }
        this.quantity = quantity;
        this.price = price;
    }
}
```

#Example 3:

For this example, let's do something a little different. Rather than handing you an explanation, we'll let you try to figure out exactly what's going on. See the two examples above for inspiration, then try to explain this one in your own language in the space provided beneath the IDE.

#Extra Resources

Here's a walkthrough that shows you how to throw custom exceptions:

https://youtu.be/OIozDnGYqIU

#Problem Statement

Sometimes we may want to throw a custom exception based on understanding a project, domain, or some business logic. For example, if we have an application that asks a user to provide a file name ending with `.txt`, an exception can be thrown if the given file name doesn't have the extension `.txt`

Of course, we want a meaningful explanation for the exception so that anyone can understand when it is thrown. In this case, we can throw a `FileExtensionException`, and give it some appropriate text.

Let's make it happen.

#Steps to be followed:

1. Create a class called `FileExtensionException` which extends the `Exception` class.
2. Give `FileExtensionException` a parameterized constructor – `FileExtensionException(String message).`
3. Perform below steps within `CustomException.java` class
    1. Define a `Scanner` instance.
    2. Prompt the user to enter a file name with the correct extension using

        ```
        System.out.println("Enter the file name with correct extension i.e. .txt ");
        ```

    3. Read the file name from the user's input.
    4. Store the filename into a String variable called `fileName`
    5. Define a `try-catch-finally` block to handle potential exceptions that might be thrown during the process
    6. Define private static `validateFileExtension()` method which takes the `fileName` as parameter and add `throws FileExtensionException` in the method signature
    7. Within the method, check If `fileName` doesn't end with ".txt", `throw new FileExtensionException("File doesn't have .txt extension").`
    8. Come back to the main method , within the try block, if the file extension is valid, you print a success message using`System.out.println("Correct file name with extension .txt").`
    9. In the catch block If an exception is caught, you print an error message using`System.out.println("Error: " + e.getMessage()).`
    10. In the `finally` block, close the scanner instance.

```
import java.util.Scanner;
```

```
public class CustomException {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.println("Enter the file name with correct extension i.e
. .txt ");
            String fileName = scanner.nextLine();
            validateFileExtension(fileName);
            System.out.println("Correct file name with extension .txt");
        } catch (FileExtensionException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static void validateFileExtension(String fileName) throws FileExte
nsionException {
        if (!fileName.endsWith(".txt")) {
            throw new FileExtensionException("File doesn't have .txt extension
");
        }
    }
}
public class FileExtensionException extends Exception {
    public FileExtensionException(String message) {
        super(message);
    }
}
```