Pro Dev Session

Pro Dev Session

You may want to write this down.

Collaboration is an essential part of being a great teammate and developer. To be a good collaborator, we must master our collaboration tools, namely git and GitHub.

Today we will get more practice with git. We'll be building on this for the next several lessons.

Pro Dev Session

You may want to write this down.

Git is a version control software that runs on your computer. You can think of it like Microsoft Word.

When you work on a project in a repository on your computer, git is running in the background.

Git essentially just runs idley behind the scenes until you tell it to do something.

The command `git add -A` tells git to monitor all the files in your project for changes.

At this point git is just watching and making note of any changes. It's not saving them.

Pro Dev Session

You may want to write this down.

The command `git commit -m "<meaningful message>"` saves the project in its current state.

This is where the Microsoft Word metaphor breaks down. When you save a document in Word, you override any previous versions.

With git, the `commit` command is similar to save but it saves a totally new version of your project exactly as it is in that moment. Old versions are preserved and never automatically overridden.

Git Practice

Work individually but with the support of your classmates to complete all of the goals below.

Goals:

Let's see how powerful version control really is!

- Create a new repository on GitHub and clone it to your local machine, using the command `git clone <address>`
- In the new repository, create a new IntelliJ project that simply prints your name to the console.
- In the terminal type `git add -A` followed by `git commit -m "initial commit"`.
- Delete the print statement and replace it with 2 integers. Then print the sum of the integers.
- In the terminal type `git add -A` followed by `git commit -m "replaces name with sum"`.
- Type `git log`. You will see a list of all your commits. Copy the long identifier next to the word commit from your initial commit. This is called a commit hash, id, or SHA (yep it has 3 names).
- Type `git checkout <SHA>`. Look in IntelliJ. All the code from your original commit is

Check-in Time

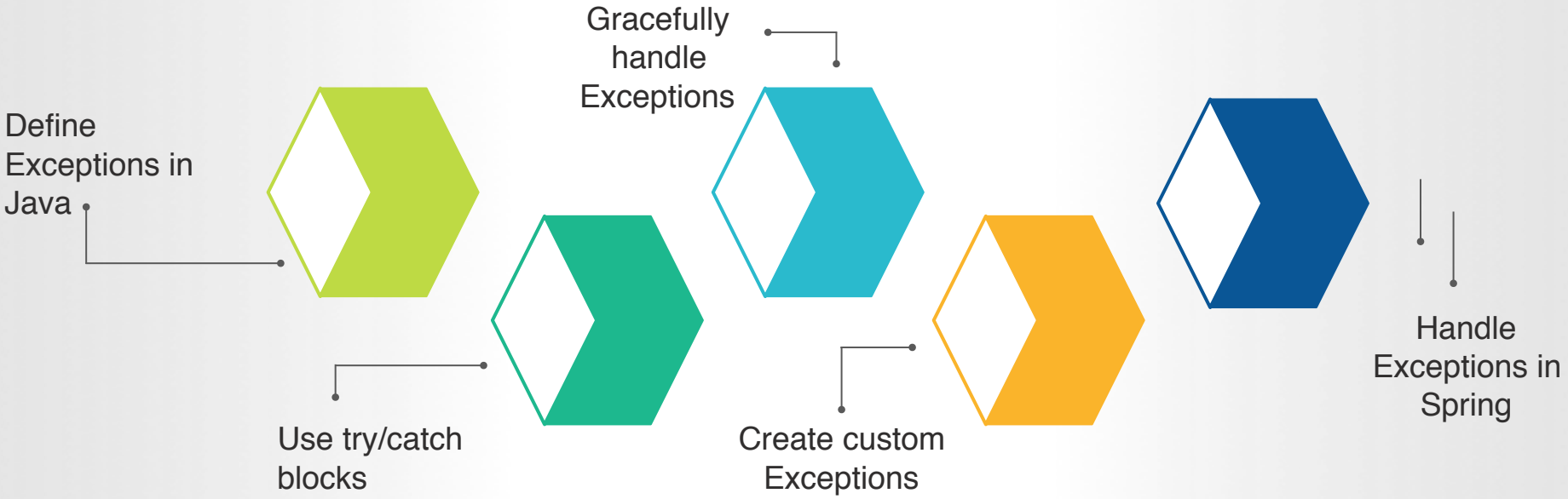
- When might this need to checkout old versions of your code be useful?
- How often do you think you should commit your code?
- What do you should do once you have commited your final change on a feature? Should you continue to only store it locally? How can you store it somewhere safer?

Stand Up!

Objectives & Key Outcomes

THE TAKEAWAYS FROM THIS CLASS

By the end of class today, you will be able to:



Objectives &Key Outcomes

THE TAKEAWAYS FROM THIS CLASS

By the end of class today, you will be able to:

Let’s Do This!

Warm-up

Warm Up

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work together but INDEPENDENTLY write your own code to complete all of the goals below.

Goals:

- Use Spring Initializr (start.spring.io) to create a new Spring Boot project
- Create a class called Note with properties id, body, and author.
- Create a RESTful API that has a GET and POST route that get and post Notes from a Note ArrayList.

15 minutes!

Stay Seated & Take 3 Deep Breaths.

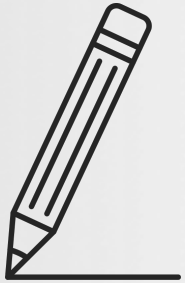
RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes. We'll start back shortly.

Handling Exceptions

Handling Exceptions

WHAT TO DO WHEN THE UNEXPECTED HAPPENS



Notebooks Ready? It's time for a brief lecture.

Handling Exceptions

WHAT TO DO WHEN THE UNEXPECTED HAPPENS

Errors occur in all programs.

Every programming language has a way to handle errors.

Exceptions are the approach that Java takes.

Handling Exceptions

WHAT TO DO WHEN THE UNEXPECTED HAPPENS

An exception happens when a problem arises during a programs execution.

By default, exceptions stop the execution of our code. This is typically not desirable.

Instead of allowing exceptions to crash our application, we generally want to handle them.

Handling Exceptions

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

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

         int a = createInt("9");
                  System.out.println(a);
   }

         public static int createInt(String x)
{
                  int y = Integer.parseInt(x);
                  return y;
         }
}
```

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

         int a = createInt("hi");
                  System.out.println(a);
   }

         public static int createInt(String x
{
                  int y = Integer.parseInt(x)
                  return y;
         }
}
```

Handling Exceptions

WHAT TO DO WHEN THE UNEXPECTED HAPPENS

Exceptions can be a handled in two ways:

1. Use a try/catch block to catch errors and handle them.
2. Pass the buck to the caller of the method to handle.

Handling Exceptions

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

          int a = createInt("hi");
     System.out.println(a);
  }

  public static int createInt(String x){
          int y;

          try {
            y = Integer.parseInt(x);
          }catch (NumberFormatException e){
            y = -1;
          }

            return y;
  }
}
```

There are pros and cons to this method.

Pro: The method always returns a valid integer.

Con: The method returns -1 for “cow”, “hi”, “five”, or any other String not parsable to an integer. Is this an expected behavior??..

Well, maybe. Plenty of languages have common methods that return -1 to represent “does not exist” or unknown values. But this could be an unexpected response and lead to bugs.

Handling Exceptions

```
public static void main(String[] args){

  try {
     createInt("hi");
  }catch(NumberFormatException e){
     System.out.println("Error: " + e);
  }

}

public static int createInt(String x){
 int y;

 try{
    y = Integer.parseInt(x);
 } catch(NumberFormatException e){
    throw new NumberFormatException("supplied value " + x + " cannot be converted to an integer");
 }

 return y;
}
```

This approach allows 2 things:

1. We can add a helpful error message.

2. The caller of createInt can handle the error however it likes. It can log the error and continue execution, halt execution, prompt the user for a new value, or do anything else it wants.

Handling Exceptions

```
public static void main(String[] args){

  try {
     createInt("hi");
  }catch(NumberFormatException e){
     System.out.println("Error: " + e);
  }

}

public static int createInt(String x){
  int y;
  y = Integer.parseInt(x);
  return y;
}
```

Note that the caller can handle the error without the try/catch in the createInt method. The try/catch in the createInt method simply allows us to add a custom message and extra code that executes when an error is caught, if needed.

Handling Exceptions

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

- Create an application that prompts a user for a number and then converts the input to integer.
- If the input is not parsable to an integer and throws a NumberFormatException, prompt the user to enter a new number. Continue this indefinitely until the user inputs a value that is parsable to an integer.

*Hint: you will need a try/catch block and a loop.
Think about which kinds of loop runs indefinitely until a condition is met.

15 minutes!

Handling Exceptions

WHAT TO DO WHEN THE UNEXPECTED HAPPENS

There are two types of exceptions in Java: Checked and Unchecked. Checked exceptions are exceptions whose handling can be verified at compile time.

When you read and write to files the file that you are reading from or writing to must exist. This a common problem and the compiler checks to verify that you are handling this exception. Pausing execution with Thread.sleep can cause another common checked exception - InterruptedException. Working with a SQL database also can throw a checked exception.

NumberFormatException is an unchecked exception, meaning the compiler does not force you to handle it. Unchecked exceptions are typically the result of bad programming or bad user input rather than interacting with outside sources like fileIO or a SQL database.

Handling Exceptions

```
public static void main(String[] args) {
        System.out.println("Start");
        pause();
        System.out.println("End");
}

public static void pause() {
    Thread.sleep(5000);
}
```

Note that the InterruptedException is a checked exception and must be handled.

Handling Exceptions

```
public static void main(String[] args) {
        System.out.println("Start");
        pause();
        System.out.println("End");
}

public static void pause() {
        try {
                Thread.sleep(5000);
        } catch(InterruptedException e) {
                System.out.println(e);
        }

}
```

Note that the InterruptedException is a checked exception and must be handled.

We can handle it with a try/catch.

Handling Exceptions

```
public static void main(String[] args) {
        System.out.println("Start");
        try {
                pause();
        } catch(InterruptedException e) {
                System.out.println(e);
        }

        System.out.println("End");
}

public static void pause() throws InterruptedException {
                Thread.sleep(5000);
}
```

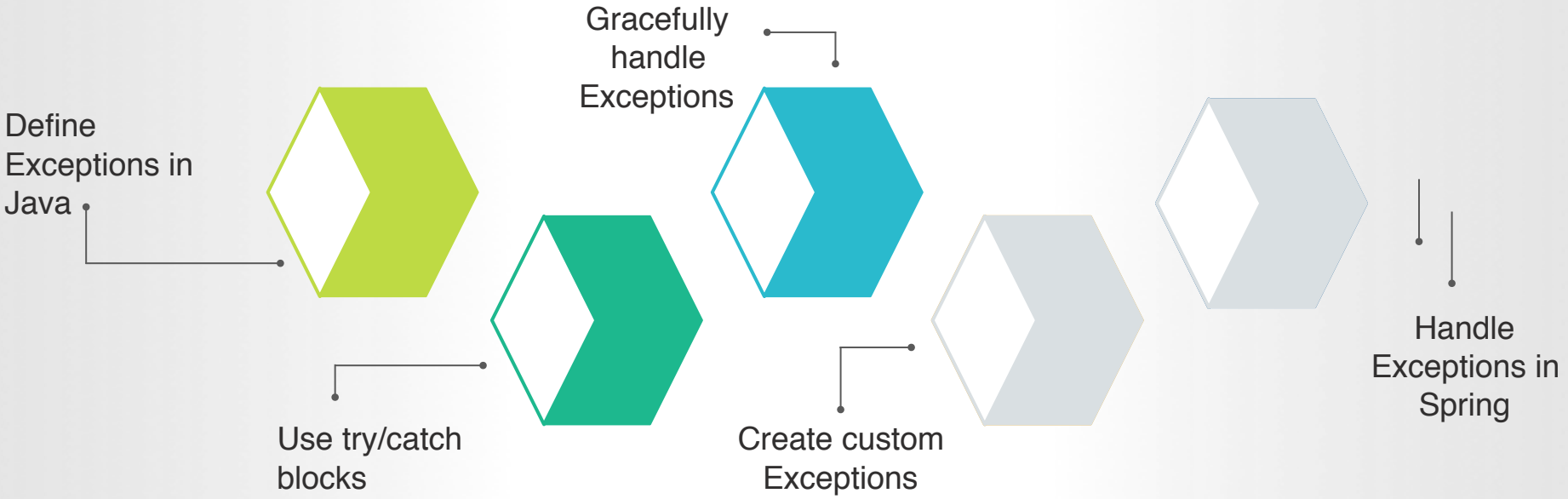
Or we can force the caller to handle it by using the throws keyword. This only works for checked exceptions.

Again, each style has its benefits and the proper approach depends on the application on the potential ramifications of an error.

Objectives & Key Outcomes

THE TAKEAWAYS FROM THIS CLASS

By the end of class today, you will be able to:

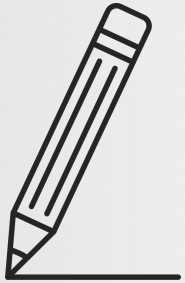


lunch.

Custom Exceptions

Custom Exceptions

WHEN BUILT IN EXCEPTIONS JUST CAN'T CUT IT



Notebooks Ready? It's time for a brief lecture.

Custom Exceptions

WHEN BUILT IN EXCEPTIONS JUST CAN'T CUT IT

Not only can we handle exceptions thrown by other methods like Thread.sleep and Integer.parseInt, but we can throw exceptions ourselves.

There are many checked and unchecked exceptions in Java, but there is sometimes a need to create unique exceptions for a given application.

In practice, you should only create custom exception if your custom exception provides utility that standard exceptions do not, like custom properties for storing application at the time of the error or methods that help handle the error.

Custom Exceptions

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
public static void main(String[] args) {
        imgToBytes("happy.jpg");
}

public static byte[] imgToBytes(String img){

    if(!img.matches(".*(\\.jpg|\\.png)")){
      throw new IllegalArgumentException("valid image extensions include .jpg and .png");
    }
    return convertToByteArray(img);

}
```

This is an example of throwing a standard exception

Custom Exceptions

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

```
public static void main(String[] args) {
        imgToBytes("happy.jpg");
}

public static byte[] imgToBytes(String img){

    if(!img.matches(".*(\\.jpg|\\.png)")){
        throw new IllegalImageExtension("valid image extensions include .jpg and .png");
    }
    return convertToByteArray(img);

}
```

This is an example of an application that may need a custom exception.

Custom Exceptions

WATCH & LEARN

Close your laptop. Eyes on my screen. Pay attention.

Checked Exception

```
public class IllegalImageExtension extends Exception {
    public IllegalImageExtension(String msg) {
        super(msg);
    }
}
```

Unchecked Exception

```
public class IllegalImageExtension extends RuntimeException {
    public IllegalImageExtension(String msg) {
        super(msg);
    }
}
```

Note: We would only create a custom exception if there were additional properties or methods that provided an advantage.

Custom Exceptions

INDEPENDENT PRACTICE

It's time to fly. Focus. Work hard. Ask for help when you need it.

Work in PAIRS to complete all of the goals below.

Goals:

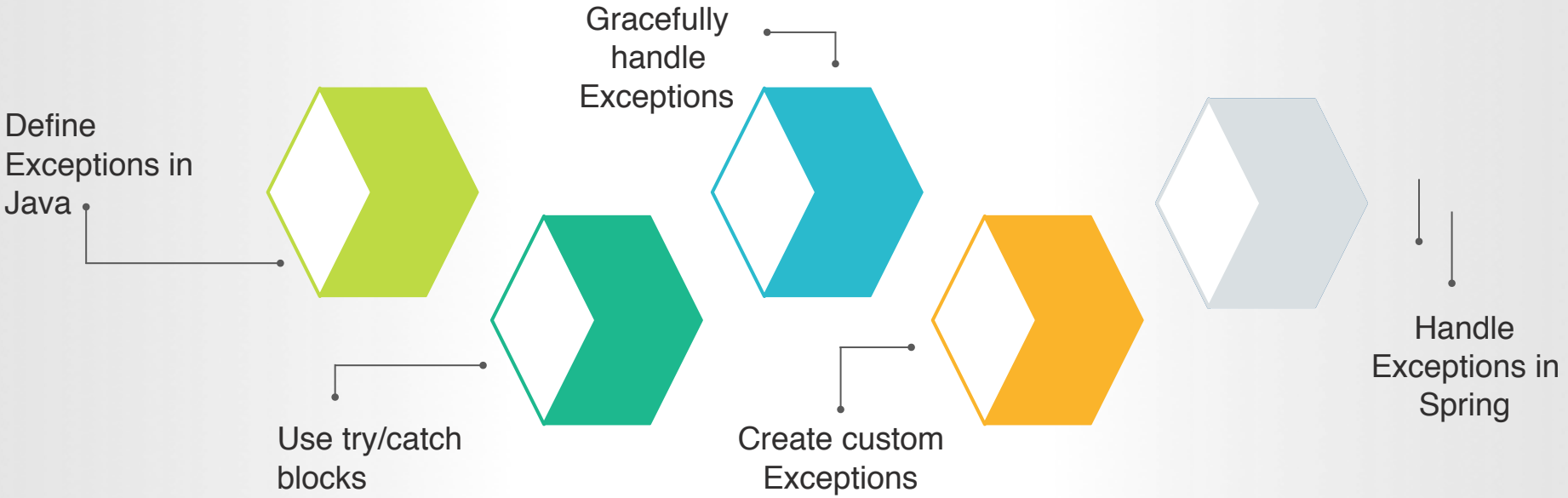
- Create an application that prompts a user for a full name
- If there is no space in the name throw a custom checked exception called IncorrectNameFormatException
- Use a try/catch to handle the error gracefully.

15 minutes!

Objectives & Key Outcomes

THE TAKEAWAYS FROM THIS CLASS

By the end of class today, you will be able to:



Stay Seated & Take 3 Deep Breaths.

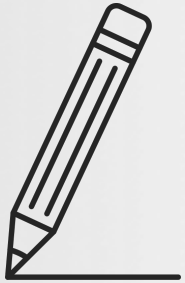
RELAX.

Now take a short walk. Clear your head. After a few minutes break, quickly review your notes. We'll start back shortly.

Exceptions in Spring

Exceptions in Spring

REST CONTROLLERS AND INPUT ISSUES



Notebooks Ready? It's time for a brief lecture.

Exceptions in Spring

REST CONTROLLERS AND INPUT ISSUES

REST controllers are prone to errors due to the reliance on user/client data.

Often times those errors are caused by unmet validation constraints but they may also be exceptions explicitly thrown or custom exceptions.

Perhaps a user is posting photos to an online image site. Internally the route may check for a valid extension and then convert the image to a byte array for storage - just like the example we saw when we were creating custom exceptions!

Exceptions in Spring

CODE-A-LONG

Open your laptop. Code with me. Don’t jump ahead.

```
@RestController
public class EchoRangeServiceController {

    @RequestMapping(value = "/echo/{input}", method = RequestMethod.GET)
    @ResponseStatus(value = HttpStatus.OK)
    public int echo(@PathVariable int input) {
        if (input < 1 || input > 10) {
            throw new IllegalArgumentException("Input must be between 1 and 10.");
        }

        return input;
    }
}
```

Exceptions in Spring

REST CONTROLLERS AND INPUT ISSUES

The framework does a pretty good job communicating errors but we can create global error handlers for all exceptions thrown by a controller method.

Exceptions in Spring

```
@RestControllerAdvice
@RequestMapping(produces = "application/vnd.error+json")
public class ControllerExceptionHandler {

 @ExceptionHandler(value = {IllegalArgumentException.class})
 @ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
 public ResponseEntity<VndErrors> outOfRangeException(IllegalArgumentException e, WebRequest request) {
    VndErrors error = new VndErrors(request.toString(), e.getMessage());
    ResponseEntity<VndErrors> responseEntity = new ResponseEntity<>(error, HttpStatus.UNPROCESSABLE_ENTITY);
    return responseEntity;
   }
}
```

Exceptions in Spring

REST CONTROLLERS AND INPUT ISSUES

There's a special way we handle exceptions thrown by JSR 303 validation constraints.

There's a lot of code here, but it's mostly boilerplate that you can copy paste.

While this may seem intimidating, learning to work with unknown code is a crucial skill for all developers.

Exceptions in Spring

```
public class Motorcycle {

    @NotEmpty(message = "You must supply a value for VIN.")
    @Size(min = 5, max = 5, message = "VIN must be 5 characters in length.")
    private String vin;
    @NotEmpty(message = "You must supply a value for make.")
    private String make;
    @NotEmpty(message = "You must supply a value for model.")
    private String model;
    @NotEmpty(message = "You must supply a value for year.")
    @Size(min = 4, max = 4, message = "Year must be 4 digits.")
    private String year;
    @NotEmpty(message = "You must supply a value for color.")
    private String color;

    //getters and setters omitted for brevity…
  }
```

Exceptions in Spring

```
@RequestMapping(value = "/motorcycles", method = RequestMethod.POST)
@ResponseStatus(value = HttpStatus.CREATED)
public Motorcycle createMotorcycle(@RequestBody @Valid Motorcycle motorcycle) {
          motoList.add(motorcycle);
          return motorcycle;
}
```

Exceptions in Spring

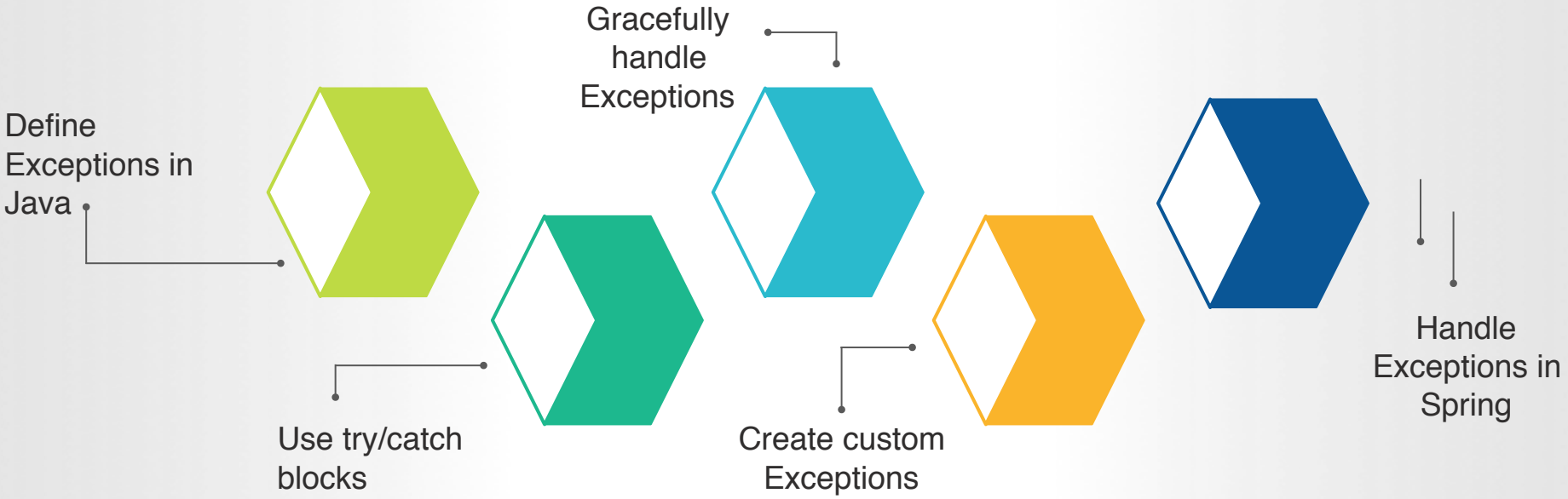
Open the code slacked to you and follow along as we add comments to each line.

Remember you can usually copy paste most of this code, but you still need to understand generally what it does.

Objectives & Key Outcomes

THE TAKEAWAYS FROM THIS CLASS

By the end of class today, you will be able to:



Wrap Up

Module 3 Lesson 3

HOMEWORK

You don’t have to submit your nightly homework, but you are expected to complete it.

For homework tonight, students should read this article on Handling Exceptions in Java

Daily Assessment

Work INDIVIDUALLY to complete all of the goals below.

Goals:

- Create a simple Post route that takes in a string from the client.
- Throw an IllegalArgumentException if the string does not start with the Letter H (case insensitive).