Set 08 Answers

1. Consider the following class:

```
public class Tornado {
   private static final int MAX_STRENGTH = 7;
   public String name = _Bill_;
   private void causeDamage() {
      // imagine code here
   }
   public static int calculateTrajectory() {
      // imagine code here
   }
}
```

Below, assume that the variable `_t_` refers to a Tornado object. The table below has rows corresponding to various expressions related to the Tornado class. The columns list various places where one might consider using those expressions. Clearly place a **_V_** in any box where the corresponding expression is ***valid syntax*** when used in the corresponding context.

	static method of Tornado class	non-static method of Tornado class	method of some other class
`name`		V	
`MAX_STRENGTH`	V	V	
`causeDamage()`		V	
`calculateTrajectory()`	V	V	
`t.name`	V	V	V
`t.MAX_STRENGTH`	V	V	
`t.causeDamage()`	V	V	
`t.calculateTrajectory()`	V	V	V
`Tornado.name`			
`Tornado.MAX_STRENGTH`	V	V	
`Tornado.causeDamage()`			
`Tornado.calculateTrajectory()`	V	V	V

2. Where can you use a _continue_ statement?

Inside of any loop.

3. Describe how a continue statement behaves in a while loop.

Goes to top of loop immediately and checks the boolean condition to see if looping should continue.

4. Describe how a continue statement works in a for-loop.

It will execute the statement that is usually processed at the end of an iteration through the loop, and then check the boolean condition to see if looping should continue.

5. Where can you use a _break_ statement?

Either in a switch statement or in a loop. (We will learn about "switch" statements later.)

6. Describe how a break statement behaves.

Causes the inner-most loop or switch statement to be exited immediately.

7. Write some faulty code that generates a null-pointer exception, catch the exception immediately and print out _exception caught_ in your catch block.

```
String s = null;
try {
        int x = s.length();   // throws exception
} catch (NullPointerException e) {
        System.out.println("exception caught");
}
```

8. Write some code that prompts the user for a numerical value, and reads their input into an `int` variable. Run the program, and try entering some text (like _cat_) instead of a number. Notice what kind of exception is thrown. Now modify your program so that it catches this exception, and instead of crashing the program, have it tell the user that he/she must enter a NUMBER, and then prompt them for input again.

```
Scanner s = new Scanner(System.in);
boolean goodNumberEntered = false;
```

```
while (true) {
    System.out.println("enter a number: ");
    try {
       int n = s.nextInt();
    } catch(java.util.InputMismatchException e) {
        s.nextLine();  //removes previous input from stream
        System.out.println(_No, you MUST ENTER A NUMBER!_);
        continue;
    }
    break;
}
```

9. Write a method called `smallSum` that takes two int parameters, `x` and `y`. If the absolute value of the sum of the integers is more than 100, throw an ArithmeticException, passing the String _I don_t like big numbers_ to the constructor of the exception. If the sum is less than 100, then return the sum. Write a quick driver to test out your method. After making sure everything works correctly, modify the driver so that it catches the exception and prints out the message that was passed to the exception_s constructor, but doesn_t crash the program.

```
public static int smallSum(int x, int y) {
    if (Math.abs(x + y) > 100) {
       throw new ArithmeticException("I don_t like big numbers!");
    }
    return x + y;
}


public void static main(String[] args) {
    try {
    smallSum(50, 60);
  } catch (ArithmeticException e) {
    System.out.println(e.getMessage());
  }
}
```

10. Explain the relationship between exception handling and the call stack. What happens if an exception is thrown but not caught anywhere in your program?

When an exception is thrown, Java looks for an appropriate _catch_ block in the place where the problem occurred. If a catch is not found there, then Java discards the current frame on the call stack, and looks for an appropriate catch block in the previous frame. If none is found, that frame is discarded, etc. If the exception is not caught anywhere, then the program terminates, and the exception propagates to the _outside world_, which for us is the Eclipse IDE. Eclipse displays that red and blue error notification with stack trace in the console.

11. Under what circumstances will the finally block run?

The finally block ALWAYS runs.