Name:	Period:

AP Computer Science A

Exceptions

Index

Exceptions

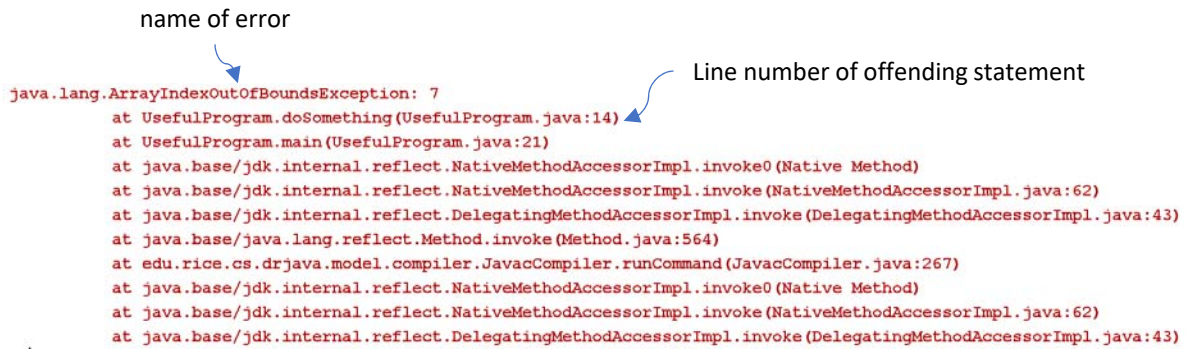
Programs don't always work correctly. When something goes wrong, Java will throw an Exception. There are two types of Exceptions, runtime and compile time.

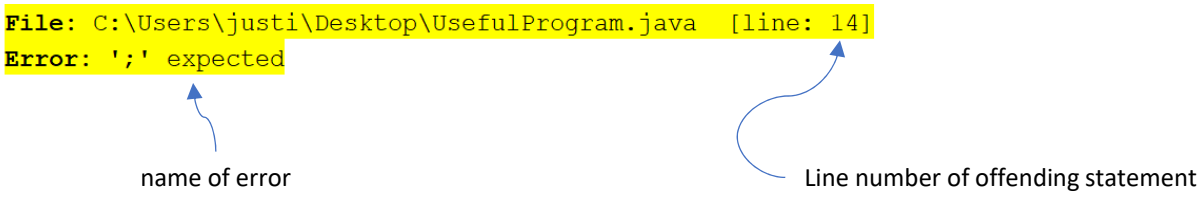
Compile Time Exceptions
Compile time exceptions are problems with your code that Java notices when you compile your code. These are usually syntax errors and typos. You must fix these issues before your code will compile successfully.

Runtime Exceptions
Runtime exceptions are problems with your code that don't cause any trouble until your program is actually running. These are usually errors where you code *could* work if some variables had different values. These include IndexOutOfBounds and NullPointer exceptions.

Infinite Loops
Another common coding problem is infinite loops. Unfortunately, these do not throw a compile time nor a runtime exception. You'll notice an infinite loop because your program will seem to have crashed (it isn't doing anything), but it is still running.
A common reason for an infinite loop is putting a semi-colon at the end of a while loop header.

Stack Trace
When Java throws an exception, it prints a Stack Trace to the console. The stack trace gives you useful information about what went wrong and the line number of the offending code. It also shows you the path Java traveled to get to the line of code that threw the exception.





Compile Time Errors

These errors are caught when you compile your program. These are often typos. You must fix all of the compile time errors before your program will run.

Occasionally your program will generate more than 1 compile time error. Always attempt to fix the first (top most) error and recompile. Sometimes the first error can make it look like many other things are broken, but when that error is fixed all the other errors go away too!

‘;’ expected

Java thinks you forgot a semi-colon at the end of a statement, or inside a for-loop header. You will also see this error if you forgot a + operator between two Strings when concatenating. This error can also occur if you forgot the { after a method header.

Notes:

reached end of file while parsing

Your curly braces don’t match up. For every { you have, you need a matching }. Make sure you are formatting your code correctly because that will help you identify where the missing curly brace is.

Notes:

‘{‘ expected

You forgot the { after the class header.

Notes:

class, interface, or enum expected

Something is wrong with your class header *or* you wrote code outside of the class. Make sure your class header looks like this:

```
public class nameOfClass
{
}
```

And make sure nothing comes before or after the class except import statements and there is no code after the class body closes.

Notes:

`class NameOfClass is public, should be declared in a file named NameofClass.java`

The name of your class must match the name of the file *exactly.* Make sure spelling and capitalization are correct.

Notes:

`cannot find symbol`
`symbol: variable nameOfVariable`
`location: class nameOfClass`

You used a variable without declaring its type. The stack trace tells you the name of the variable you used, the class it was used in, and the line number. Make sure you spelled the variable correctly when you used it *and* when you declared it. Make sure you have access to the variable where you are using it (did you declare the variable in a different method than you are using it?) Sometimes this happens if you are declaring and initializing a variable in one statement, but you forgot to give your variable a name!

`cannot find symbol`
`symbol: method nameOfMethod`
`location: variable nameOfVariable of type type`

You called a method that does not exist. Check the spelling of the method name where you called it *and* in the method header. Make sure the object you are calling the method on actually declares that method.

`cannot find symbol`
`symbol: class nameOfClass`
`location: class nameOfAnotherClass`

You declared a variable of a type that doesn't exist. Check the spelling of your variable declaration and your class name. Is the class saved? Is it saved in the correct place?

Notes:

`incompatible types: type cannot be converted to otherType`

You are assigning a value to an incorrectly typed variable. For example, you are trying to assign a String to an int variable. You can also get this error if you call a `void` method and try to assign its return value to a variable (void methods do not return anything)

Notes:

`unclosed string literal`

You are missing a ” at the end of a String.

Notes:

`This class does not have a static void main method accepting String[]`

You tried to **run** your program without selecting the class that includes the **main** method. The main method is the starting point of a Java program, so you can only *run* classes that contain a main method.

Notes:

`Non-static method methodName() cannot be referenced from a static context`

You tried to **run** your program without selecting the class that includes the **main** method. The main method is the starting point of a Java program, so you can only *run* classes that contain a main method.

Notes:

Non-static method *methodName*() cannot be referenced from a static context

You tried calling a non-static method from inside a static method. **DO NOT MAKE METHODS STATIC TO FIX THIS!** This commonly happens when you try to call a method from within the `main` method. The main method is not actually part of the object, so it doesn't have access to the methods. What object do you want to *do* that behavior? You need to use dot-notation to call a non-static method on an object. You may need to create an instance of an object first, so that you have a reference to it.

Another reason this error can occur is because you forgot to use the `new` keyword when instantiating an object.

Notes:

Non-static variable *variableName* cannot be referenced from a static context

You tried accessing an instance variable from a static method. **DO NOT MAKE INSTANCE VARIABLES STATIC TO FIX THIS!** This commonly happens when you try to access an instance variable from within the `main` method. The main method is not actually part of the object, so it doesn't have access to the instance variables. You can only access instance variables from the regular (non-static) methods. You probably want to create an instance of the object and call one of its methods that will manipulate the instance variable appropriately.

Notes:

Invalid method declaration; return type required

Your method header is not correctly formatted. Don't forget the return type of the method. If the method does not return anything, then use void as the return type. If you are trying to write a constructor (which does not have a return type in its header) then make sure you have spelled the method name exactly the same as the class name.

Notes:

Missing return statement

The method header says that this method returns some value, but it doesn't have a return statement in the method. Sometimes you get this error even when you do have a return statement in your method. That happens because there exists some path through your method that does not execute the return statement. Usually this happens when your return statement is inside an if-statement or a loop. What does this method return if that if-statement is false or if the loop never iterates because its condition was false before it started?

Notes:

<identifier> expected

Make sure your method name does NOT contain any spaces. Alternatively, something is wrong in the parameter list of your method header. Each parameter needs to have a type and a variable name.

Notes:

Method *methodName* in class *ClassName* cannot be applied to given types;
required: *type,type*
found: *type*
reason: actual and formal argument lists differ in length

You called a method but did not pass the correct number of arguments. Double check that your argument types match the number and types of the method's parameters. Make sure you are calling the correct method. If you have multiple classes with the same method names, but different parameter lists, make sure you are calling the method on the correct type of object.

Notes:

`illegal start of type`

You probably are writing code inside of a class, but not inside of a method. Double check your curly braces, and make sure all your statements are wrapped in a method body. The only code that can be outside a method body is an instance variable declaration.

Notes:

`illegal start of expression`

You may be trying to write a method inside of another method. That's a no-no. Check your curly braces and tabbing. Confirm that all of your methods are lined up inside the class.

Notes:

`bad operand types for binary operator '+'`

The only types of data that can be *added* are ints, doubles, and Strings. The '+' symbol doesn't work for anything else.

Notes:

`Missing method body, or declare abstract`

There is probably a semi-colon at the end of your method header. You probably don't want there to be a semi-colon at the end of your method header.

If you are writing an abstract method in an abstract class, then you *don't* want a body and you *do* want a semi-colon. However, you also need to add the `abstract` keyword to your method header.

Notes:

Array required, but *type* found

You are using array syntax (square brackets) with something that is not an array. Commonly you are tying to use array syntax with a List, but just as often you forgot the declare your variable as an array.

Notes:

int cannot be dereferenced
double cannot be dereferenced
boolean cannot be dereferenced

Primitive values do not have methods. You cannot call a method on a primitive variable (int, double, or boolean)

Notes:

'else' without an 'if'

The else keyword *must* follow an if-statement block. If there if *any* code in between the if-statement block and the else-statement then you will get this error.
Also, the lone else statement must be the LAST part of an if/else if/else chain. This error could mean that you have an if/else/else if chain.

Notes:

no suitable constructor found

The argument list you passed to a constructor did not match any of that object's constructor's parameter lists. Double check the types of the arguments you are passing and double check the constructor's parameter list.

Notes:

`incompatible types: possible lossy conversion from double to int`

You are trying to assign a double value to an int variable. Java will not let you do this because it has to truncate the fractional part of the double to fit it in an int variable, which means the value might lose some precision.

You can force Java to make this assignment by *casting* the double into an int before assigning it to the int variable.

```
double pi = 3.14;
int closeEnough = (int)pi;    // evaluates to 3
```

Notes:

Runtime Errors

Runtime errors occur when your program is running. Usually these occur because the data that is stored in a variable is not exactly what you though it would be.

You can use your editor's debugging features to step through your code and watch how certain variables change to determine what went wrong. Or you can sprinkle `System.out.println()` statements throughout your code to figure out how variables are changing.

`NullPointerException`

If you don't assign a value to an Object variable, then it defaults to the value `null`. Null objects are not objects at all. Null has no behavior. Null is not an object, it is nothing. You cannot call a method on *nothing*. Look for a statement like:

```
obj.methodName()
```

Double check that the variable `obj` has been assigned a non-null value.

You may have a "constructor" that is supposed to be initializing the variable that is giving you a NullPointerException. Are you sure that is a constructor and not just a method with the same name as the class? If the method header has a return type then it is not a constructor and those variables have not been initialized!

Notes:

`ArrayIndexOutOfBoundsException: #`

An element was attempted to be accessed from an array using an index that is too big or too small (negative). The error message ends with a number. That number is the out of bounds index. Fix your code so that the index doesn't go out of bounds.
Sometimes the index is fine, but the size of the array is wrong. Make sure your array was initialized to the correct length. If you used a variable to initialize the length of the array, make sure the value of that variable was not 0.

Notes:

`StringIndexOutOfBoundsException: begin #, end #, length #`

The arguments to the `substring` method fell out of bounds of the string. The error message tells you the length of the string as well as the start and end indexes (exclusive)..

Notes:

`StackOverflowError`

You wrote a method that calls itself, which then calls itself again, and again, and again...

Or you wrote a constructor that creates an instance of another object of the same type (that calls the same constructor again, which creates another instance, which creates another object, etc…)

This error is harder to track down. The line number given in the stack trace is not necessarily where the error is occurring, but where Java ran out of memory. You need to work your way backwards through the code, find the recursive method call, and fix it.

Notes:

Other “Errors”

There are some other common errors that can cause your program to work incorrectly, but since the code is correct Java does not see them as errors. Ie, your code “failed successfully”.

Semi-colon at the end of an if-statement or while-loop

Do not put a semi-colon at the end of an if-statement, else-statement, for-loop, or while-loop. The semi-colon *detaches* the body from the header. The following two code segments are equivalent due to the semi-colon in the first one.

```
while(condition);
{
  // do something
}
```

```
while(condition)
{

}
// do something
```

The symptoms of this bug are different depending on the type of control structure you're using. A while loop will end up in an infinite loop, a for-loop will seem to execute its body exactly once, or give an error about an undefined variable if you are using the counter inside the body. An if-statement will always execute its body, regardless of how it's condition evaluates.

Notes:

No curly braces around the body of an if-statement or loop (while, for, for-each)

You can get away without using curly brackets to enclose the body of an if-statement or loop as long as the body has *exactly* one statement in it (and that statement is *not* a variable declaration) This will be a bug if you expect to have more than one statement in your body. Consider the following two code segments which are equivalent.

```
if( x < 10 )
  x = 100;
  System.out.println(x);
```

```
if( x < 10 )
{
  x = 100;
}
System.out.println(x);
```

Notice the first segment is meant to only print the value of x if it was less than 10. However, it actually works like the second block which always prints the value of x, no matter what its value was.

Symptoms of this bug are that code that you think should not execute because a condition is false is always executing. Or, in the case of a for-loop, it seems like the loop only ever iterates once.

Notes:

Constructors with return values are not constructors

Constructors are special types of methods and they have two rules. Rule #1: Constructors do not have return types. Rule #2: Constructors are named the same as the class they are in.

Rule #1 is the distinguishing factor between a constructor and a regular method. If there is a return type, then it is a regular method. If there isn't, then it is a constructor. Confusingly, a regular method *can* following Rule #2.

If you write a "constructor" but you accidentally give it a return type, then you have actually created a regular method with a confusing name.

Consider the following two classes. One of them has a constructor, the other does not.

```
public class Foo
{
  public Foo()
  {
    // this is a constructor
  }
}
```

```
public class Foo
{
  public void Foo()
  {
    // this is NOT a constructor
  }
}
```

If your constructor has parameters, then you will likely get a compile-time error telling you that the specified constructor does not exist.

However, because Java automatically inserts a default constructor (with no parameters) if you do not specify any constructors, your code may compile if you accidentally give a return type to a constructor with no parameters. Common symptoms of this bug are that all of your instance variables are being initialized to their zeroish values even though your "constructor" is initializing them to something else.

Notes: