**HADIAH 1**

**Pemrograman Berorientasi Objek**



**Helen Natalia**

**51019010**

**STMIK KHARISMA MAKASSAR**

1. **50 pesan error di java (error tentang apa berikan penjelasan).**
2. **“… Expected”**

Kesalahan ini terjadi ketika ada sesuatu yang hilang dari kode. Seringkali ini dibuat oleh titik koma yang hilang atau tanda kurung tutup.

1. **“Unclosed String Literal”**

Pesan kesalahan “unclosed string literal” dibuat ketika string literal berakhir tanpa tanda kutip dan pesan akan muncul pada baris yang sama dengan kesalahan. (@DreamInCode) Literal adalah kode sumber dari suatu nilai

Biasanya, ini terjadi ketika:

* String literal tidak diakhiri dengan tanda kutip. Ini mudah diperbaiki dengan

menutup string literal dengan tanda kutip yang diperlukan.

* String literal melampaui garis. Literal string panjang dapat dipecah menjadi beberapa literal dan digabungkan dengan tanda tambah (“+”).
* Tanda kutip yang merupakan bagian dari string literal tidak diloloskan dengan garis miring terbalik (“\”).

1. **“Illegal Start of an Expression”**

Ada banyak alasan mengapa kesalahan " Illegal Start of an Expression " terjadi. Ini menjadi salah satu pesan kesalahan yang kurang membantu. Beberapa developer mengatakan itu disebabkan oleh kode yang buruk.

1. **“Cannot Find Symbol”**

Ini adalah masalah yang sangat umum karena semua identifiers di Java perlu dideklarasikan sebelum digunakan. Ketika kode sedang dikompilasi, kompiler tidak mengerti apa arti pengenal.

Ada banyak alasan Anda mungkin menerima pesan " Cannot Find Symbol ":

* Ejaan pengenal saat dideklarasikan mungkin tidak sama dengan saat digunakan dalam kode.
* Variabel tidak pernah dideklarasikan.
* Variabel tidak digunakan dalam lingkup yang sama dengan yang dideklarasikan.
* Kelas tidak diimpor.

1. **“Public Class XXX Should Be in File”**

Pesan " Public Class XXX Should Be in File " terjadi ketika kelas XXX dan nama file program Java tidak cocok. Kode hanya akan dikompilasi ketika kelas dan file Java sama.

1. **“Incompatible Types”**

" Incompatible Types " adalah kesalahan dalam logika yang terjadi ketika pernyataan penugasan mencoba memasangkan variabel dengan ekspresi tipe. Itu sering muncul ketika kode mencoba menempatkan string teks ke dalam bilangan bulat — atau sebaliknya. Ini bukan kesalahan sintaks Java.

1. **“Invalid Method Declaration; Return Type Required”**

Pesan kesalahan perangkat lunak Java ini berarti tipe pengembalian suatu metode tidak secara eksplisit dinyatakan dalam signature metode.

Ada beberapa cara untuk memicu “deklarasi metode tidak valid; jenis pengembalian diperlukan” kesalahan:

* Lupa sebutkan jenisnya
* Jika metode tidak mengembalikan nilai maka "void" perlu dinyatakan sebagai jenis dalam tanda tangan metode.
* Nama konstruktor tidak perlu menyatakan tipe. Tetapi jika ada kesalahan dalam nama konstruktor, maka kompiler akan memperlakukan konstruktor sebagai metode tanpa tipe yang disebutkan.

1. **“Method <X> in Class <Y> Cannot Be Applied to Given Types”**

Pesan kesalahan perangkat lunak Java ini adalah salah satu pesan kesalahan yang lebih bermanfaat. Ini menjelaskan bagaimana signature metode memanggil parameter yang salah.

Metode yang dipanggil mengharapkan argumen tertentu yang didefinisikan dalam deklarasi metode. Periksa deklarasi metode dan panggil dengan hati-hati untuk memastikan mereka kompatibel.

1. **“Missing Return Statement”**

Pesan " Missing Return Statement” terjadi ketika suatu metode tidak memiliki pernyataan pengembalian. Setiap metode yang mengembalikan nilai (tipe non-void) harus memiliki pernyataan yang secara harfiah mengembalikan nilai itu sehingga dapat dipanggil di luar metode.

Ada beberapa alasan mengapa kompiler menampilkan pesan "pernyataan pengembalian yang hilang":

* Pernyataan pengembalian dihilangkan begitu saja karena kesalahan.
* Metode tidak mengembalikan nilai apa pun tetapi tipe void tidak dideklarasikan dalam tanda tangan metode.

1. **“Possible Loss of Precision”**

" Possible Loss of Precision " terjadi ketika lebih banyak informasi diberikan ke variabel daripada yang dapat disimpannya. Jika ini terjadi, potongan akan dibuang. Jika ini baik-baik saja, maka kode tersebut perlu secara eksplisit mendeklarasikan variabel sebagai tipe baru.

Kesalahan "kemungkinan hilangnya presisi" biasanya terjadi ketika:

* Mencoba menetapkan bilangan real ke variabel dengan tipe data integer.
* Mencoba menetapkan double ke variabel dengan tipe data integer.

1. **“Reached End of File While Parsing”**

Pesan kesalahan ini biasanya terjadi di Java ketika program tidak memiliki kurung kurawal penutup (“}”). Terkadang dapat dengan cepat diperbaiki dengan menempatkannya di akhir kode.

1. **“Unreachable Statement”**

" Unreachable Statement " terjadi ketika pernyataan ditulis di tempat yang mencegahnya dieksekusi. Biasanya, ini setelah pernyataan break atau return.

1. **“Variable <X> Might Not Have Been Initialized”**

Ini terjadi ketika variabel lokal yang dideklarasikan dalam suatu metode belum diinisialisasi. Itu bisa terjadi ketika variabel tanpa nilai awal adalah bagian dari pernyataan if.

1. **“Operator ... Cannot be Applied to <X>”**

Masalah ini terjadi saat operator digunakan untuk jenis/tipe, bukan dalam definisinya.

Ini sering terjadi ketika kode Java mencoba menggunakan string tipe dalam perhitungan. Untuk memperbaikinya, string perlu dikonversi ke integer atau float.

1. **“Inconvertible Types”**

Kesalahan " Inconvertible Types”" terjadi ketika kode Java mencoba melakukan konversi ilegal.

Misalnya, boolean tidak dapat dikonversi ke bilangan bulat

1. **“Missing Return Value”**

Anda akan mendapatkan pesan " Missing Return Value " ketika pernyataan pengembalian menyertakan jenis yang salah.

Biasanya, ada pernyataan pengembalian yang tidak mengembalikan apa pun.

1. **“Cannot Return a Value From Method Whose Result Type Is Void”**

Kesalahan Java ini terjadi ketika metode batal mencoba mengembalikan nilai apa pun,

Seringkali ini diperbaiki dengan cara mengubah ke signature method untuk mencocokkan jenis dalam pernyataan kembali. Dalam hal ini, instance void dapat diubah .

1. **“Non-Static Variable ... Cannot Be Referenced From a Static Context”**

Kesalahan ini terjadi ketika kompiler mencoba mengakses variabel non-statis dari metode statis.

Untuk memperbaiki "variabel non-statis . . . tidak dapat direferensikan dari kesalahan konteks statis, coba dua hal ini:

* Deklarasikan variabel sebagai statis di signature.
* Periksa kode karena dapat membuat instance objek non-statis dalam metode statis.

1. **“Non-Static Method ... Cannot Be Referenced From a Static Context”**

Masalah ini terjadi saat kode Java mencoba memanggil metode non-statis di kelas non-statis.

Memanggil metode non-statis dari metode statis berarti mendeklarasikan turunan dari kelas yang memanggil metode non-statis.

1. **“(array) <X> Not Initialized”**

Anda akan mendapatkan pesan “(array) <X> not initialized” ketika sebuah array telah dideklarasikan tetapi tidak diinisialisasi. Array tetap panjangnya sehingga setiap array perlu diinisialisasi dengan panjang yang diinginkan.

1. **“StringIndexOutOfBoundsException”**

Ini adalah masalah yang terjadi saat kode mencoba mengakses bagian dari string yang tidak berada dalam batas string. Biasanya, ini terjadi ketika kode mencoba membuat substring dari string yang panjangnya tidak sama dengan parameter.

Seperti indeks array, indeks string dimulai dari nol. Saat mengindeks string, karakter terakhir kurang dari satu panjang string. Pesan kesalahan perangkat lunak Java "StringIndexOutOfBoundsException" biasanya berarti indeks mencoba mengakses karakter yang tidak ada.

1. **“NullPointerException”**

"NullPointerException" akan terjadi ketika program mencoba menggunakan referensi objek yang tidak memiliki nilai yang ditetapkan padanya

Program Java sering memunculkan exception ketika:

* Pernyataan mereferensikan objek dengan nilai nol.
* Mencoba mengakses kelas yang ditentukan tetapi tidak diberi referensi.

1. **“NoClassDefFoundError”**

"NoClassDefFoundError" akan terjadi ketika interpreter tidak dapat menemukan file yang berisi kelas dengan main method .

Ini bisa terjadi jika:

* File tidak berada di direktori yang benar.
* Nama class tidak sama dengan nama file (tanpa ekstensi file). Juga, nama-nama itu peka huruf besar-kecil.

1. **“NoSuchMethodFoundError”**

Pesan kesalahan ini akan terjadi ketika perangkat lunak Java mencoba memanggil metode kelas dan metode tersebut tidak lagi memiliki definisi

Error: Could not find or load main class wiki.java

Seringkali kesalahan perangkat lunak Java "NoSuchMethodFoundError" terjadi ketika ada kesalahan ketik dalam deklarasi.

1. **“NoSuchProviderException”**

“NoSuchProviderException” terjadi ketika penyedia keamanan diminta yang tidak tersedia:

javax.mail.NoSuchProviderException

Saat mencoba menemukan mengapa "NoSuchProviderException" terjadi, periksa:

* Konfigurasi JRE.
* Java\_home diatur dalam konfigurasi.
* Lingkungan Java yang digunakan.
* Entri penyedia keamanan.

1. **AccessControlException**

"AccessControlException" menunjukkan bahwa akses yang diminta ke sumber daya sistem seperti sistem file atau jaringan ditolak, seperti dalam contoh ini

ERROR Could not register mbeans java.security.

AccessControlException: WFSM000001: Permission check failed (permission “(“javax.management.MBeanPermission” “org.apache.logging.log4j.core.jmx.LoggerContextAdmin#-

[org.apache.logging.log4j2:type=51634f]” “registerMBean“)” in code source “(vfs:/C:/wildfly-10.0.0.Final/standalone/deployments/mySampleSecurityApp.war/WEB-INF/lib/log4j-core-2.5.

jar )” of “null”)

1. **“ArrayStoreException”**

Sebuah "ArrayStoreException" terjadi ketika aturan elemen casting dalam array Java rusak. Berhati-hatilah dengan nilai apa yang Anda tempatkan di dalam array.

Ketika sebuah array diinisialisasi, jenis objek yang diperbolehkan ke dalam array perlu dideklarasikan. Kemudian setiap elemen array harus memiliki tipe objek yang sama.

1. **“bad magic number”**

Pesan kesalahan perangkat lunak Java ini berarti ada yang salah dengan file definisi kelas di jaringan. Berikut ini contoh dari Sisi Server

Pesan kesalahan " bad magic number”" terjadi ketika:

* Empat byte pertama dari file kelas bukanlah bilangan heksadesimal CAFEBABE.
* File kelas diunggah seperti dalam mode ASCII, bukan mode biner.
* Program java dijalankan sebelum dikompilasi.

1. **“broken pipe”**

Pesan kesalahan ini mengacu pada aliran data dari file atau soket jaringan yang telah berhenti bekerja atau ditutup dari ujung lainnya

Penyebab kesalahan "pipa rusak" sering kali meliputi:

* Kehabisan ruang awal disk.
* RAM mungkin tersumbat.
* Aliran data mungkin rusak.
* Proses membaca pipa mungkin telah ditutup.

1. **“could not create Java Virtual Machine”**

Pesan kesalahan Java ini biasanya terjadi ketika kode mencoba memanggil Java dengan argumen yang salah

1. **“Console** “

Metode untuk mengakses console berbasis karakter, jika bisa, diasosiasikan dengan mesin virtual Java saat ini.

1. **“File”**

Sebuah representasi abstrak dari pathname file dan direktori.

1. **“Writer”**

Kelas abstrak untuk menulis kedalam aliran karakter.

1. **“Reader”**

Kelas abstrak untuk membaca aliran karakter.

1. “**CharConversionException”**

Kelas dasar untuk pengecualian konversi karakter.

1. **“EOFException”**

Memberikan signal jika akhir file atau aliran telah dicapai secara tak terduga ketika input.

1. **“FileNotFoundException”**

Memberikan signal bahwa sebuah percobaan untuk membuka file yang telah ditentukan pathname-nya gagal.

1. **“IOException “**

Menandakan bahwa pengecualian I/O dari beberapa jenis telah terjadi.

1. **“NotActiveException “**

Muncul ketika serialisasi atau deserialisasi tidak aktif.

1. **“NotSerializableException”**

Muncul ketika sebuah contoh memerlukan interface serialisasi.

1. **“UnsupportedEncodingException”**

Enkoding karakter tidak didukung.

1. **“IOError “**

Muncul ketika ada error I/O yang serius telah terjadi.

1. **“BufferedOutputStream”**

Mengimplementasikan sebuah aliran output buffer.

1. **”BufferedReader** “

Membaca teks dari aliran output karakter, menyangga karakter untuk menyediakan pembacaan karakter, array, dan lines yang efisien.

1. **“BufferedWriter”**

Menulis teks ke aliran output karakter, menyangga karakter untuk menyediakan penulisan karakter, array, dan lines yang efisien.

1. **“FilenameFilter”**

Contoh kelas yang mengimplementasikan antarmuka ini digunakan untuk memfilter nama file.

1. “Flushable “

Merupakan destinasi dari data yang dapat di-flush

1. **“Object Input”**

ObjectInput memperluas interface dari DataInput untuk menyertakan pembacaan objek

1. **“ObjectInputValidation “**

Panggil balik interface yang memperbolehkan validasi dari objek didalam grafik

1. **“Serializable”**

Serialisasi dari sebuah kelas diaktfkan oleh kelas yang mengimplementasikan interface Serializable java.io

1. **30 file header java (import java.io. ? ), (file header itu fungsinya untuk apa ?).**

* Interface Summary

1. Closeable = Closeable adalah sumber atau destinasi dari data yang dapat di tutup
2. DataInput = Interfacenya menyediakan pembacaan byte dari aliran biner dan merekonstruksi dari data dalam semua tipe primitive Java
3. DataOutput = Interfacenya menyediakan pengkonversian data dari semua tipe primitive Java ke kumpulan seri dari byte dan menuliskannya ke dalam aliran biner
4. Externalizable = Externalizable hanya menyimpan identitas kelas dan menuliskannya ke dalam serialization stream, dan merupakan tanggung jawab dari kelas untuk menyimpan dan memulihkan konten dari contohnya
5. FileFilter = Sebuah filter untuk pathname yang abstrak
6. FilenameFilter = Contoh kelas yang mengimplementasikan antarmuka ini digunakan untuk memfilter nama file.
7. Flushable = Merupakan destinasi dari data yang dapat di-flush
8. Object Input = ObjectInput memperluas interface dari DataInput untuk menyertakan pembacaan objek
9. ObjectInputValidation = Panggil balik interface yang memperbolehkan validasi dari objek didalam grafik
10. ObjectOutput = ObjectOutput memperluas interface dari DataOutput untuk menyertakan penulisan objek
11. ObjectStreamConstants =Konstanta dituliskan ke dalam Object Serialization Stream
12. Serializable = Serialisasi dari sebuah kelas diaktfkan oleh kelas yang mengimplementasikan interface Serializable java.io

* Class Summary

1. BufferedInputStream = Menambahkan fungsionalitas ke aliran input lain -yaitu, kemampuan untuk menyangga input dan mendukung metode penandaan dan reset.
2. BufferedOutputStream = Mengimplementasikan sebuah aliran output buffer.
3. BufferedReader = Membaca teks dari aliran output karakter, menyangga karakter untuk menyediakan pembacaan karakter, array, dan lines yang efisien.
4. BufferedWriter = Menulis teks ke aliran output karakter, menyangga karakter untuk menyediakan penulisan karakter, array, dan lines yang efisien.
5. CharArrayReader = Mengimplementasikan sebuah aliran output dimana datanya dapat ditulis ke dalam byte array.
6. CharArrayWriter = Mengimplementasikan penyanggaan karakter yang dapat digunakan sebagai Writer.
7. Console = Metode untuk mengakses console berbasis karakter, jika bisa, diasosiasikan dengan mesin virtual Java saat ini.
8. File = Sebuah representasi abstrak dari pathname file dan direktori.
9. Writer = Kelas abstrak untuk menulis kedalam aliran karakter.
10. Reader = Kelas abstrak untuk membaca aliran karakter.

* Exception Summary

1. CharConversionException = Kelas dasar untuk pengecualian konversi karakter.
2. EOFException = Memberikan signal jika akhir file atau aliran telah dicapai secara tak terduga ketika input.
3. FileNotFoundException = Memberikan signal bahwa sebuah percobaan untuk membuka file yang telah ditentukan pathname-nya gagal.
4. IOException = Menandakan bahwa pengecualian I/O dari beberapa jenis telah terjadi.
5. NotActiveException = Muncul ketika serialisasi atau deserialisasi tidak aktif.
6. NotSerializableException = Muncul ketika sebuah contoh memerlukan interface serialisasi.
7. UnsupportedEncodingException = Enkoding karakter tidak didukung.

* Error Summary

1. IOError = Muncul ketika ada error I/O yang serius telah terjadi.
2. **Format penulisan Bahasa pemrograman java.**
   * + Harus disimpan berekstensi \*.java
     + Nama file harus sama dengan nama class public, misalnya nama filenya myHello.java, maka nama class public juga harus myHello

Contoh:

public class myHello {

//harus sama dengan nama file (myHello.java)

}

* + - Comment sebaiknya sebaiknya ditulis untuk menjelaskan sebuah class atau method.   
      Komentar *single line*, menggunakan tanda //  
      Format : // teks

Komentar blok, diawali dengan tanda /\* dan diakhiri dengan tanda \*/  
Format : /\*  
\* teks  
\*/

komentar dokumentasi, ini adalah komentar spesial yang mengacu pada komentar *javadoc*. Komentar ini dapat diekstrak melalui file HTML menggunakan perintah *javadoc* pada JDK.  
Komentar ini diawali dengan tanda /\*\* dan diakhiri dengan tanda \*/. Komentar ini bertujuan untuk menjelaskan mengenai keseluruhan *class*, *method*, konstruktor, *interfaces*, dan *fields*.  
Format : /\*\*  
\* teks  
\*/

* + - Java statement adalah suatu baris yang diakhiri dengan titik koma.

Contoh:

System.out.println(“Hello this is my first java”);

* + - Block adalah satu atau beberapa statement yang berada diantara tanda kurung kurawal { dan diakhiri dengan kurung kurawal }.

Contoh:

public static void main(String[]args){

System.out.println(“Hello this is my first java”);

System.out.println(“Java is very good”);

}

* + - Case Sensitive

Java bersifat Case Sensitive, artinya huruf besar atau kapital dan huruf kecil dibedakan.

Contoh:

String nama = "Petani Kode";

String Nama = "petanikode";

String NAMA = "Petanikode.com";

System.out.println(nama);

System.out.println(Nama);

System.out.println(NAMA);

Tiga variabel tersebut merupakan tiga variabel yang bebeda, mesikipun sama-sama bernam nama. Banyak pemula yang sering salah pada hal ini. Karena tidak bisa membedakan mana variabel yang menggunakan huruf besar dan mana yang menggunakan huruf kecil. Apabila kita membuat variabel seperti ini:

String jenisKelamin = "Laki-laki";

Maka kita harus memanggilnya seperti ini:

System.out.println(jenisKelamin);

**Bukan seperti ini:**

System.out.println(jeniskelamin);

Perhatikan, huruf K adalah huruf kapital.

* Gaya Penulisan Case

Gaya penulisan case (case style) yang digunakan oleh Java adalah: camelCase, PascalCase, dan ALL UPPER.

Gaya penulisan camelCase digunakan pada nama variabel, nama objek, dan nama method.

Contoh:

String namaSaya = "Dian";

Lalu untuk PascalCase digunakan pada penulisan nama class.

Contoh:

class HelloWOrld {

//...

}

Perhatikan nama class-nya, kita menggunakan huruf kapital di awal, dan huruf kapital pada huruf W untuk memisahkan dua suku kata.

Sedangkan camelCase huruf depannya menggunakan huruf kecil, dan awalan suku kata berikutnya menggunakan huruf besar.

// ini camelCase

belajarJava

// ini PascalCase

BelajarJava

Lalu, penulisan ALL UPPER atau semaunya kapital digunakan pada pembuatan nama konstanta.

Contoh:

public final String DB\_NAME = "petanikode";

Untuk penulisan dua suku kata atau lebih, ALL UPPER dipisah dengan garus bawah atau underscore (\_).

* Statement dan Ekspresi pada Java

Statement dan eksrepsi adalah bagian terkecil dalam program. Setiap statement dan ekspresi di Java, harus diakhiri dengan titik koma (;).

Contoh statemen dan ekspresi:

System.out.println("Hello World");

System.out.println("Apa kabar?");

var x = 3;

var y = 8;

var z = x + y;

Statemen dan ekspresi akan menjadi instruksi yang akan dikerjakan oleh komputer.

Pada contoh di atas, kita menyuruh komputer untuk menampilkan teks "Hello World", dan "Apa kabar?". Lalu kita menyuruhnya untuk menghitung nilai x + y.