Cheatography

Core Java Cheat Sheet

by evanescesn09 via cheatography.com/88543/cs/20288/

OOP

Encapsulation	The process of binding related classes, objects and operations together is called Encapsulation	Using access modifiers, packages
Abstraction	The process of specifying what to do without specifying how to do it	Using abstract classes and interfaces
Inheritance	When one class inherits the properties of another class	Using Aggregation, Composition
Polymorphism	Same thing is done in different ways	Using compile-time and run-time polymorphism

Encapsulation

default	accessible to classes only in the same package
public	accessible to all classes in any package
private	accessible to only a specific method, or class
protected	accessible to classes in the same package, and sub-classes of this class

Abstraction

Abstract Class	When a class has one or more unimplemented methods, it should be declared abstract. The sub-classes should provide implementation for the unimplemented methods, else they should also be declared abstract	Used: When default implementation is needed for some methods, and specific implementations for other methods based on the class implementing them

Abstraction (cont)

Interface	Blueprint of a class. It contains only static, final variables and only unimplemented methods. Classes implement the interface should implement all methods of the interface, or be declared abstract	Used: to support multiple inheritance

Abstract classes don't support multiple inheritance, whereas interfaces do

Inheritance

Aggregation	When one class contains a reference of another class	Loosely coupled classes
Association	When one class is made up of another class	Tightly coupled classes

Java does't support multiple inheritance directly, it supports it only via Interfaces

Polymorphism

Compile-time	Also called overloading. When methods have same name but different signature (return-type, number of parameters, type of parameters etc)
Run-time	Also called overriding. When child-classes over-write method implementations of parent-class.

static keyword

static field	Shared by all members of the class. It can be accessed before objects are created
static method	Can be accessed without creating an instance of the class. They can only access static variables and static methods. Cannot access this or super

static keyword (cont)

static block	Used when some computation is to be done to initialize the static variables. This block is executed once when the class is initially loaded into memory
static class	We cannot declare top-level classes as static. Only inner classes can be static. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class

final

fields	treated as constants
methods	cannot be overridden by child classes
classes	cannot be inherited

finalize( )

finalize() method is a protected and non-static method of java.l-ang.Object class. This method will be available in all objects you create in java. This method is used to perform some final operations or clean up operations on an object before it is removed from the memory

String Creation

Literal : String s = " "	Creates Strings in String pool, in JVM. Multiple strings can have same value. Only one copy of the word exists in the String pool, and the references of it are updated.
Object: String s = new String( );	Creates a string object in heap. The heap in-turn checks the JVM String Pool to see if there exists a string with same value.

```
String s1 = "abc";
String s2 = "abc";
s1 == s2 returns true;
======================
String s1 = new String("abc");
String s2 = new String("abc");
s1 == s2 returns false;
But s1.equals(s2) returns true;
```

String Immutability

Strings in java are immutable because changing the value of a String literal changes the value of other Strings that reference the literal, which leads to inconsistency in the program. To prevent this, strings in java are immutable.

Storing passwords in Strings

It is best to store passwords as char[ ] because if passwords are stored as Strings, the string tends to be in the JVM pool even after all references to it no longer exist. This causes a vulnerability in the system. In case of Char[ ], once all the references to char[ ] are gone, the Java Garbage Collector deletes the char[ ] to preserve memory. So, it's safer.

StringBuilder, StringBuffer

StringBuilder	To create mutable strings in Java
StringBuffer	To create thread-safe mutable strings in Java

String methods

s.charAt(int index)
s.compareTo(s2), s.compareToIgnoreCase(s2)
s.concat(s2)
s.contains(sequence of characters)
s.equals(s2), s.equalsIgnoreCase(s2)
s.length()
s.replace(character, replacement))
s.replaceAll(character, replacement)
s.subString(int startIndex)
s.subString(int startIndex, int endIndex)
s.toUpperCase( ), s.toLowerCase( )
s.toCharArray()
s.trim( )
String s = String.valueOf(int, or long or double)
String[] s1 = s.split(String regex)
String[] s1 = s.split(String regex, int limit)

StringBuffer, StribgBuilder methods
s.append(s2)
s.deleteCharAt(int index)
s.indexOf(string), s.indexOf(string, fromIndex)
s.insert(int index, objectValue)
s.replace(int startIndex, int endIndex, String)
s.reverse( )
s.toString( )
s.trimToSize( )
s.setCharAt(int index, charSequence)