Cheatography

Java Methods and encapsulations Cheat Sheet

by Jianmin Feng (taotao) via cheatography.com/79308/cs/19357/

Accessors

private	class private
(default)	package private
protected	package private+subclass
public	any class

Optional specifier

static	class method,accessed by class name or obj reference
abstract	w/o body, even empty {}
final	no overridden by subclass
synchr-onized	OCP, thread safe
native	interact with other language
strictfp	float point calc portable

Return type

Required	void =return; or no return statement
void method(){return;}	void method(){}
String method(){return "";}	String method(){-return null;}
String[] method(){-return null;}	Integer method(){-return null;}
void method(){return null;}	This not compiled
String method(){if ... return null; else return;}	Not compile

Method name

follow variable naming rule

start with	letters, &, _
can contains	number

Can not start with number

Parameter list

Must have a parameter list

if no parameter, use empty ()

parameter separated by ;

Optional Exception List

throws excaption_name1,exception_name-2,...

Method body

MUST for all non abstract method, Could by empty {}

abstract method could not have a body, even empty {} one

Order must be followed

accessor --optional-modifier -- return type name() optional-throws .... {}

accessor could be omitted for default access

accessor,return type, name, (), {} are required

signature: return type + name + ()

overloading: same name, but differs in parameter list, overloading won't care about return type

varargs

```
//varargs=variable arguments,
//array but variable in length
//only last one parameter could be varargs
//accessed like array
public void walk (int start, int...steps)
{ System.out.print(steps.length +"," +steps[0]);}
```

varargs (cont)

```
walk(1);//ArrayIndexOutOfBound-sException
walk(1,2) //1,2
walk(1,2,3);//2,2
walk(1,null);//NullPointExc-eption
walk(1,new int[] {1,2,3}); //3,1
walk(1,new int[3]); //3,0
```

protected members

pacakage	subclass	direct access	by obj ref
same	Yes	yes	yes
same	No	yes	Yes
diff	Yes	yes	No *
diff	diff	No	No

could access protected member from parent by obj ref of subclass itself

Design static methods and fields

2 use cases

shared component among instances	util/helper which not require any obj state

2 calling approaches

by class name	by obj ref (null obj can too)
Static variables	Counter, Constant
Static initializer-run once	static {....}//bad practice

import static java.util.Arrays.asList

import static java.util.Arrays.*;

static import java.util.Arrays.*;	syntax error
import static java.util.Arrays;	Can not static import class

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Design static methods and fields (cont)

Name conflicts: not compiled	Solution: regular import,ref by classname

Object has its status, but shared the code, and static data

Static vs instance

type	calling	legal	how
static method	static	Yes	Class name
static method	instance	No	create obj
instancd method	static	Yes	class name,obj ref
instance method	instance	Yes	obj ref

Object has its status, but shared the code, and static data

Passing data among method

pass by value	change in callee not affect caller
pass by ref	change in callee do affect caller

Return value, if not used, will be discarded.

Overloading a method

Method signature	name+parameter list
Overloading:	same name+different parameters
varargs	2nd test can't compile, varargs and array = same parameter list

Overloading a method (cont)

	void test(int... a){}; void test(int[] a){}
	test(new {1,2,3}) call either
	test(1,2,3) call test(int... a) only
autoboxing	void fly(int a){}; void fly(Integer a){};
	fly(3) call fly(int a); if not exit, call fly(Integer a)
Ref type	void fly(Object o){};
	fly(3):match fly(int) >fly(Integer) >fly(Object o)
Primitive	autocast(wider), explicitly cast(narrower)

overloading matching order

exact match -> unwrapping -> promotion -> wrapping ->varargs(exact match, promotion, wrapper)

promotion is an independent check, it retains no influence over wrapping check

void play(Long l){}

play(4); compile error, 2 step conversion, only 1 conversion allowed, play(4L) will work.

Create constructors

Same name as class	no return type	
default no-argument constructor	auto generated if no constructor provided	
overloading of constructor	Constructor chain	by this(...)

Create constructors (cont)

Calling Constructor	This(...): not in static method	first line of method
	new Constructor(...)	Create new object
final field	must be initialized by the time of constructor completion	
super()	call automatically in any constructor implicitly	
	all class are subclass of Object	
order of initialization	super() is always call in every constructor implicitly; super(...) must be called explicitly	

super() ->static declaration and static {} as its order ->instance declaration and {} as its order ->constructor

example

class Bunny{ }	public bunny(){}	wrong
	public Bunny(){}	OK
	public void bunny(){}	OK
	public void Bunny(){}	wrong

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Encapsulating data

Encapsulation: binding fields with method; data hiding(hiding properties+implementation details): private fields+public setter and getter; purpose: data validation and integrity + flexibility code for upgrade and maintenance

private attributes	public getter(), setter(), isA()
Immutable class	Math, String
Class w/o public setter	no status change of object once created
safe to passing around and easy to maintain	bette performance by limiting the number of copies (string pool)
String and StringBuffer	Mutated by StringBuffer, return a String
JavaNean	
a class with no-argument constructor	naming convention:instance variable, getA,setA,isA,
reusable	easy to code

Lambdas expression

Why lambdas?

less code	delayed implementation	declarative

what's lambdas?	a declarative block of code

passed to the associated 1 method interface, as delayed dynamic implement the method

Syntax

parameter	->	body

Lambdas expression (cont)

()->true	simplest format
a->a.canHop()	type is optonal
(Animal a) -> {return a.canHop();}	can't miss ; or return if use{}; need () if has type
(a,b)->a.canHop()	() required if >1args; if >1 statement, need {}
Context	functional method to match the only method of an interface
	replace the implementing class of the interface
	parameter type will be get from the associated interface API

Notes

lambdas expr could access static/instance variable. method parameter and local variable also fine if not assigned a new value

can not redeclare a variable same name as local variable

(a,b)->{int a=0;return 5;}	won't compile

Lambdas example - Predicate

```
//Predicate interface
public interface Predicate<T>{
    boolean test(T t); }
//
class Animal{
  private String canHop;
  public void setCanHop(boolean
b)
  { canHop =b; }
  public boolean canHop()
  { return canHop; }
}
public interface CheckTrait
 { boolean test(Animal a); }
public class CehckIfHopper
implements CheckTrait{
   public boolean test(Animal a)
  { return a.canHop();}
}
//compare
import java.util.function.Predi-
cate;
public class TestLambdas {
   public static void main(S-
tring[] args){
     Animal a1 = new Animal();
     a1.setCanHop(true);
     print (a1,a->a.canHop());
     // print2 (new checkIfHo-
pper());
   }
   void print(Animal a, Predi-
cate<Animal> p) {
      if (p.test(a) {} }
   void print2(Animal a,
CheckTrait c) {
      if (c.test(a) {}
}
```


Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Lambdas example - Predicate (cont)

```
//for Predicate version, no interface and
implementing class needed
// {return a.canHop();} will be delayed
dynamic implementation of test method in
Predicate interface.
```

Type must match in lambda expr, or just do put type

Lambdas - ArrayList.removeIf()

```
//Java 8 intergrated Predicate interface into
ArrayList
default boolean removeIf(Predicate<? super E>
filter)
arrayList.removeIf(s->s.charAt(0)!='h');
```

ArrayList<? super E> : super class of E, upper bound: "This can be cast to E".
ArrayList<? extends E>: hold type= subclass of E, lower bound: "E can be cast to this."
ArrayList< E>: holder type E

What happens after new Constructor()

1. allocate memory space on the heap

2. create the object and instance variables are initialized with default value

3. explicit initialize the instance variabls

4. constructor code are executed

Notes:static variable are initialized once for all objects to be created, so it is before step 2

for no argument constructor, super() is called first for initialization

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com