Hamster als Datentyp

Void rechtsUm( Hamster X)

X= Parameter ist eine Ojektvariaabele( Name, Referenz)

Vom Typ „Hamster“

Class MeinHamster extends Hamster

{

Void rechtsUm()

{

This.linksUm();

This.linksUm();

This.linksUm();

}

}

Überladen von Methoden

* Angenommen , wir wollen den Hamster bis zur nächsten Wand laufen lassen

naechsteWand();

naechstewand( int gap);

naestewand( int gap, int anzahl);

naesteWand( boolean,…)

Überladen von Methoden: die Methoden haben den gleichen Namen, Unterscheiden sich aber in der Anzahl und/oder Typ des Parameters.

naechsteWand(true);

Konstrucktoren

Methoden einer Klasse, die:

Den Namen de Klasse tragen

Die beim erzeugen neuer Objekte automatisch aufgerufen werden

Sie werden zum Initialisieren neuer Objekte verwendet

Der Entwickler der Klasse übernimmt

Verantwortung für das initialisieren der Objekte

Dass MeinHamster()

{

MeinHamster( int r, int s, int b ,int a)

{

}

Kontrucktoren:

Konstrucktoren durch eigende Methoden ersetzen

Trennen sie die Objekterzeugung von der Applikation

Klassenattribute/-methoden

Attribute sind Eigenschaften einen Objektes : Blau, Rot, Gelb

Klassenattribute sind gemeinsame Eigenschafft aller Objekte einer Klasse

Alle Objekte einer Klasse können auf diese Eigenschaft zugreifen.

Klassenatribute werden mit den Schlüsselwort static definiert

Class MeinHamster extends Hamster

{

Static int gesammelterKoerner == 0;

}

Void sammle()

{

MeinHamster.gesammelteKoenerGesamt++;

}

Klassenmethoden werden auch mit den Stichwort static deklariert

Klassenmethoden sind unabhängig von Objekt verwendbar

Der Ablauf der Klassenmethoden erfolgt über

Angabe des Klassennamens

Klasse.Methode z.b. Hamster.getStandardHamster

Klassenmehtoden dürfen nur auf (static)Klassenattribute zugreifen

Grund: eine statische Methode kann unabhägig von Objekten

Aufgerufen werden

Klasse Terrethorium

Terretorium.getAnzahlReihen() : int

Terretorium.getAnzahlSpalte(): int

Terretorium.getAnzahlHamster( r, s ) : int auf Kachel

-„-.getAnzahlHamster(): int im Gebiet

Terretorium.mauerDa(r, s) : boolean Ist da eine Mauer auf der Kachel

Terretorium.getAnzahlKoerner(r , s) int wv Körner aind auf der Kachel

Terretorium.getAnzahlKoerner() int wv Körner sind im Gebiet

Subobjekte

Die klasse Hamster betrachteen wir auch als Datentyp

Datentypen können zum Deklarieren neuer Variabelen verwendet werden

1 lokal

2 global

3 klasse

Datenfelder/ Array

Zusammenfassungen von Daten gleichen Typs

Datenfelder können gleichzeitig mehrere Werte enthalten.

Der Zugriff auf Werte im Datenfeld erfolgt elementweise

Bsp.: feld[0]=10;

Feld[10]=feld[9];

Feld[i]=i++;

feld[h]=hamster.getStandardHamster();

alle Daten eines Feldes heißen Dimension

daten reichen von Wert 0 bis Dimension -1

auch Index oder Offset genannt.

In Java müssen Datenfelder deklariert/definiert werden

Deklaration <DT> [] <Bezeichner> ;

Bekanntmachung <DT> <Bezeichner> [] ;

Definition <ArrayBezeichner> =new <DT>[Dimension];

<DT> [] <Bezeichner> = new <DT> [Dimension];

Int []< x > =new <DT>[100];

Beim Zugriff auf Datenfelder muss der Entwickler auf die Grenzen des Arrays achten

Grenzen: 0 bis Größe -1, Größe: Array.length;

Arrays von Objeckten

Mehr Dimensionale Felder

Zweidimensionales Feld als eindimensionales Feld

Funktion/Methoden können Arrays erzeugen und als Funktionswert zurückliefern

Bisher boolean rehctFrei(); boolean linksFrei();

Neu: int[] erzeuge Feld().., Hamster [] erzeuge Kaefig()

Felder als Parameter für Methoden

Void prozedur( int [] x ) boolean funktion (Hamster [] y);

BubbleSort

Sortiert von ganzen Zahlen

Objekte sind Instanzen einer Klasse

Eine Klasse ist ein Bauplan und ein Datentyp

Eine Objektvariabele ist ein Bereich für ein Objekt und referenziert Objekte

Mehrere Variabelen können ein Objekt Referenzieren

Objekte die nicht mehr Referenziert werden, werden durch den Garbage collector

Entsorgt

Polymorphie

Vielgestaltigkeit ist die Möglichkeit, dass eine Objektvariable auf Objekte „anderer“ Datentypen/Klassen referenzieren darf

Hamster paul = new Hamster();

Hamster polymorph = new MeinHamster();

In der Imperativen Programmierung verboten.

Eine Objektvariable eines Types(Klasse) kann Objekte von Subtypen(abgeleiteten Klassen) referenzieren.

Eine polymorphe Objektvariabele die also auf ein Objekt eines Subtypes referenziert

Darf dieses Objekt nur so einsetzen als wäre es ein Objekt des type der Objektvariabele

Da der Entwickler sehr genau weiß was später genau für ein Objekt referenziert wrd kann er den Compiler dies expliziet erklären.

Typumwandlung( typecast )

Hamster x = new MeinHamster();

x.rechtsUm();

((MeinHamster)x).rechtsUm();

Dynamisches Binden

Setzt man Polymorphie mit Klassenarchieven ein, in den Methoden überschreibt werden, entscheidet sich erst zur Laufzeit d. Anwendung, welche Methodenimplementierung zum Einsatz kommt

Diese Entscheidung ist abhängig von dem referenzierten Objekt, für das die Methode aufgerufen wird

Abstrakte Klassen

Sind unfertige Baupläne [es können keine unfertigen Objekte Instanziiert werden]

Unfertig bedeutet es gibt min.1 Methode die nicht implementiert ist

d.h. es fehlt der Rumpf der Methode abstract void methode();

die Klasse und die unfertigen Methoden werden durch das Schlüsselwort abstract

gekenzeichnet

da die abstrakte Klasse nicht instanziiert werden kann, wird sie ledigleich in der Vererbung eingesetzt die abgeleiteten Klassen müssen dann die abstrakten Methoden implementieren

Ergebnis ist eine fertige klasse

werden die fehlenden Methoden nicht implementiert, ist auch die abgeleitete Klasse Abstrakt

Interface

Interfaces sind Klassen , die ausschließlich aus abstrakten Methoden bestehen

Die Kennzeichnung der Methoden durch das Schlüsselwort abstract entfällt

Das Schlüsselwort class wird durch interface ersetzt

Interfaces können nicht instanziiert werden

Konkrete Klassen implementieren Interfaces

Class x implements IFace

Auch bei Interfaces und Abstrakten Klassen gilt Polymorphie( samt Eigenschaften der Protokolle/der Siegnatur) und danymisches Binden

Der Datentyp der Objektvariable ist entscheident für die Fähigkeiten für Objekte( ohne das Typecast verwendet werden müssen)

JAVA

Zugriffsrechte und Pakete

-Private: nur innerhalb der Klasse kann auf die Eigenschaft oder die Methode zugegriffen werden

Protected: Zugriff ist innerhalb der Vererbungsherrachy erlaubt

Aber in Java haben auch alle Klassen des Gleichen Paketes zugriff auf Protected Elemente

Public: öffentlicher Zugriff ist von überall aus möglich

<none> alle Klassen des gleichen paketes haben zugriff

Pakete dienen dem Strukturieren von Ozellen und dem Sauberhalten des Namensraums

Pakete / Klassen aus fremden Quellen können Importiert werden

Dazu nehmen sie die anweisung Import Paket.\*;/ import Paket.Klasse;

Paketnamen werden auf Verzeichnissstrukturen gemappt

Oakete dürfen weiter Unterteilt werden in SubPakete um einen Weltweiten austausch von Paketen zu vereinfachen verwendet mann für Paketnamen üblicherweise Domainnamen

In Java sind alle Klassen direkt oder indirekt von der Klasse Object abgeleitet

Die Klasse Onject bietet drei wichtige Methoden an, die ggf. in den eigenden Klassen überschrieben werden müssen

toString(): String liefert eine Zeichenkettenrepräsentation des Objektes

hashCode(): int liefert eine eindeutige Objekt ID

equals(): Boolean liefert eine Aussage ob zwei Objekte gleich sind

Autoboxing:

Alles wird in ein Objekt aber nur wen es unbedingt sein muss

Fals sie individuelle Ausgaben für Objekte ihrer Klasse wünschen, dann überschreiben sie die Methode toString(): String(Zeichenkette)

Die Vergleiche auf gleicheit füren bei primitiven Datentypen immer zu den „Richtigen“ Ergebniss

Der Vergleich auf gleicheit für Objekte müssen differenziert betrachtet werden

== vergleiche die Objekt referenzen

Equals() vergleicht je nach Implementierung

Wie muss equals() implementiert werden? „equals contract“

1. Jedes Objekt liefert im vergleich mit sich selbst true
2. Wenn x „==“ y true ist, dann muss auch y „==“ x true sein
3. Wenn x „==“ y true ist und y“==“z true ist dan muss auch x“==“ z true sein
4. X“==“Y muss auch nach wiederholter ausführung true/false sein
5. X“==“ null muss false sein
6. If(this == other)

Return true;

5 if(other ==null)

Return false;

(2),(3),(4)

Prüfen, ob alle eigenen Attribute mit allen korespondierenden anderen Attribute übereinstimmen

Bsp.:

This.a1 == other.a1

This.a2 == other.a2

This.aN == other.aN

Der hashCode() Contract:

Egal wie oft man HashCode aufruft die Methode liefert immer den gleichen wert

Wenn 2 Objekte gleich sind dann haben sie den gleichen HashCode

!!Wer Equals überschreibt muss auch HashCode überschreiben!!

If(x.equals(y)) 🡪 Dan🡪x.hashCode == y.hashCode

Ungleiche Objekte sollen verschiedene HashCodes haben

Class KlasseSteffen{

Public int hashCode(){

Return startwert +this.meinWert \* multiplikator

(!=0) nicht zu große primzahl

}

}

1. hashCode = 39 \* 19 + this.attribut(int)
2. hashCode = 39 \* hashCode(1) + (int)(atribut B^(atributB>>>32))
3. hashCode = 39 \* hashCode(2) + atributC.hashCode()
4. hashCode= 39 \* 19 + ((this.boolean)?1:0);

return hashCode & 0x7FFFFFFF;

Fehlerbehandlung und Exceptions

Die Exceptiones treten zur Laufzeit auf

Objekte der Klasse Exceptiones bzw. von deren Subklassen( Exceptiones sind Throwable)

Arten von Exceptions

* checked exception:
  + ausnahmen die vorhersehbar sind
  + ausnahmen durch den Programmierer Selbst behandelt werden können
* Error
  + Ausnahmen, die nicht durch den Programmierer behandelt werden können
  + Runtime exception
    - Können nicht gefangen werden
    - Programm interne ausnahmen
      * Nullpointer
      * Stack Overflow
      * Division durch 0

Ausnahmen werden behandelt indem die geworfene Ausnahme gefangen( catch) wird

Das Fangen von Ausnahmen erfolgt in einem try .. catch .. finaly Block

Try Block kapselt den kritischen Programmabschnitt, der mutmaßlich eine Ausnahme wirft

Catch-Block fängt die geworfene Ausnahme und reagiert darauf

Es dürfen mehrere Catch-Blöcke pro try-Block definiert werden

Hinweis – keine leeren Catch-Blöcke nach Möglichkeit Fehler spezifische ausnahmen fangen

Vermeide catch(Exception e)

Finaly-block : wird immer ausgeführt , sobald der try-block oder der catch-block endet

Eine Methode die eine Methode die eine Exception wirft oder weitergibt, muss dies in der Signator ausweisen

<DT> <Bezeichner(<Parameter>) throws exception, exceptionA,exception

Ungeprüfte exception: muss nicht mit catsh gefangen werden

Geprüfte Exceptions: (1) cath

(2)deklarieren, dass die Methode eine Exception wirft

Eigene Exception Klassen werden von Exception abgeleitet

Class MeineException extends Exception{

MeineException(string msg){

Super(msg);

}

}

Wozu eigene Exception Klassen?

Damit auf unterschiedliche ausnahmen unterschiedlich reagiert werden kann.

Generics

Wozu Generics?

Generics sind Typ Parameter für Klassen

Mit Generics lässt sich die konkrete Verwendung einer Klasse zu dem Zeitpunkt festlegen, wenn die Instanz der Klasse gebildet wird

Einsatz von generischen Klassen möglichst nach Angabe der Typparameter, da bei RAW-Typen verwenden der Typprüfung nicht durchführt

Typparameter dürfen auch bei der Definition von Methoden eingesetzt werden

Der Einsatz der Typparameter für Methoden sorgt wiederum für Typsicherheit

Entwicklung der Klasse fällt Selten auf den gleichen Zeitpunkt wie deren verwendung

Class Klasse boolean vergleiche(T t1, T t2){

If(t1 == t2)

Return 0;

If(t1 > t2)

Return 1;

Return -1;

}

Klasse<String> ksA = new Klasse<>();

Klasse<String> ksB = new Klasse<>();

If(ksA.vergleiche(ksB)>0)

Kann zu Problemen führen weil u.U. die Implementier und der Methode einer Klasse nicht zu den späteren verwendeten Typargumenten passt

Aus diesem grund lässt sich der Typparameter einer Klasse einschränken

Class Klasse <T extends Numbers >

Das Collection-Framework

„Rahmenwerk“ für Datenstruckturen(dynamische)

Im Collection Framework finden wir

1. Interfaces(Verhalten von Datenstruckturen )
2. Implementierungen(Klassen die die Interfaces Implementieren)
3. Algorithmen die nützliche Funktionen für Daten Sammlungen bereitstellen

Vorteile:

reduziert den Programmieraufwand

verbessert die Qualität der Software

verbessert die Zusammenarbeit mit anderen durch wohl definierte Schnittstellen

Collection Interface: die Wurzel der meisten Implementierungen

Sammlung von Objekten die als Elemente bezeichnet werden<E>

Das Set:

ist eine menge von Objekten ohne Dublikate

List:

Liste von Objekten die die Dublikate enthalten darf und sortiert ist

Queue:

Warteschlange, First-In First-Out

Wer zuerst kommt mahlt zuerst

Deque:

Duble endet queue

Warteschlange mit 2 enden

FiFo, LiFo

Map:

Schlüsselwert > wert

Key > value

Sortierte Kollektionen

Einige Kollektionen fügen die Elemente <E> auf Basis einer Sortierfolge ein

1. TreeSet
   1. TreeSet<Integer> nach zahlen
   2. TreeSet<String> Lexikographisch
   3. TreeSet<Car>

Die sortierten Kollektionen erwarten, dass die Argumente für die generischen Typparameter Klassen sich, deren Objekte sich vergleichen lassen

In diesem Sinne vergleichbar wären Klassen, die

1. Vom Comparator ableiten
2. Compareable implementieren

Compare(); --> int

Sowohl bei der Ableitung als auch bei der Implementierung dreht sich alles im die abtracten Methode compare

Int compare(obj1,obj2) 0: bei gleichheit }\ Interface

<0: falls obj1 < obj2 } > Comperator

>0: falls obj2 < obj1 }/

Das Interface Compareable erwartet die Implementierung der abtrackten Methode compareTo

-<0 :

Int obj1.compareTo(obj2) - 0 :

->0 :