- **Welchen Zweck haben Packages in Java?**

  Zur Organisation für zusammengehörige Klassen (Klassenname nur innerhalb des Packages eindeutig)

- **Wie können Packages von der JVM gefunden werden?**

Über einen Pfad im Dateisystem (CLASSPATH+Package+Subpackage+Klasse)

- **Wie viele Verzeichnisse kann die Umgebungsvariable CLASSPATH enthalten?**

Beliebig viele.

- **Gibt es Klassen im JDK, die auch ohne import-Klausel zur Verfügung stehen?**

Das Package Java.lang

- **Was versteht man unter einem ”qualifizierten Namen“?**

Ein Element eines Packages (Package-Präfix+Bezeichner)

- **Was ist der Unterschied zwischen import und static import?**

Import importiert nur Klasse (import java.lang.Math.*)
static import nur die Statischen Variablen oder Methode (PI)

- **Was besagt eine sogenannte package-Klausel?**

Zu welchem Package der Quelltext gehört

- **Was passiert, wenn in einer Quelltextdatei keine package-Klausel existiert?**

Sie gehört zum Default-Package

- **Wie können Namenskonflikte im Zusammenhang mit Packages aufgelöst werden?**

Gibt es Namenskonflikt in Importierte Klassen müssen qualifizierte Namen angegeben werden.

- **Kann ein Java-Archiv (jar-file) mehrere Packages enthalten?**

Ja in einem Zip-Archive mit festgelegter Struktur.

- **Gibt es Dateien, die in jedem jar-file vorhanden sein müssen?**

Es gibt die Datei META-INF/MANIFEST.MF

- **Was ist der Vorteil von jar-files?**

Package zusammen in einer Datei
weniger Platzbedarf da zip
digitale Signierung möglich

- **Wie können Sie sich Zugang zu einem jar-Archiv verschaffen, wenn Sie das Programm jar nicht zur Verfügung haben?**

Zip Ordner entpacken

- **Wozu dienen Interfaces?**

Dient als Schnittstelle ohne Implementierung. (Vertrag)
Eine Klasse, die ein Interface implementiert muss alle Methoden des Interfaces mit genau den vorgegebenen Parameterlisten implementieren, ansonsten ist es eine ganz gewöhnliche Klasse

- **Kann ein Interface Methoden enthalten?**

Nur die Köpfe, alles pubblic

- **Kann ein Interface Daten enthalten?**

Öffentliche Konstanten (PI)

- **Was versteht man unter dem Begriff Design by Contract?**

Interface legt „vertragliche“ Schnittstelle fest, die erst von einer Klasse implementiert werden muss.

- **Kann ein Interface private Methoden enthalten?**

Nein, nur Public

- **Wie viele Interfaces kann eine Klasse implementieren?**

Beliebig viele

- **Können von einem Interface Variablen definiert werden?**

Ja lokale Variablen

- **Wie kann man ein Interface-Objekt erzeugen?**

Interface-Typen können keine Objekte erzeugen, jedoch können der Variablen Objekte einer implementierenden Klasse zugewiesen werden

- **Was versteht man unter dynamischem Binden?**

Richtige Methode ist vom aktuellen Objekt abhängig, das wiederum ist erst zur Laufzeit bekannt, darum entscheidet JVM erst zur Laufzeit welche Methode verwendet wird.

- **Beschreiben Sie den Unterschied zwischen statischem und dynamischem Typ.**

Statischer Typ: Eine Variable wird mit einem bestimmten Typ definiert
dynamischer Typ: Der Variable wird im Laufe des Programms ein bestimmtes Objekt einer implementierenden Klasse zugewiesen

- **Muss eine Klasse auch nicht benötigte Methoden eines Interfaces implementieren?**

Es müssen ALLE Methoden eines Interfaces implementiert werden

- **Gibt es einen Unterschied, ob eine Variable vom Interface-Typ ist oder vom konkreten Typ?**

Ja, Interface Typ: es kann sein, dass mehr Methode existieren

- **Welchen Vorteil hat es, wenn der Copy-Konstruktor einer Klasse einen Interface-Typ-Parameter hat?**

Sie sind flexibler anwendbar (man hat mehr Möglichkeiten, da mehr Klassen möglich sind)

```
class Name implements Interface {
  Name (Interface Variable) {
    a = Variable.a
} }
```

Allgemeine Vorteile Interface:

- Trennung von Schnittstelle und Implementierung
- Eine Anwendung kann eine beliebige Implementierung einer Schnittstelle verwenden ohne Änderungen am Code vornehmen zu müssen.
- Auch ein späterer Austausch ist möglich.
- Konkrete Klasse ist nur bei Konstruktoraufruf notwendig.
- Interface muss für alle Anwendungsfälle passende Methoden definieren.

- **Was versteht man unter einer Superklasse?**

Die Ausgangsklasse (Basisklasse) aus der Abgeleiter wird

- **Welche Arten der Vererbung kennen Sie und was ist der Unterschied?**

Implementierung eines Interfaces (class classname implements interfacename {…})
Erweiterung / modifizierung (class classname extends baseclass {…})

- **Welche Abhilfe gibt es, wenn eine Methode unbrauchbar für die abgeleitete Klasse ist?**

Ererbte Methoden können redefiniert werden (neu implementiert werden). Jedoch mit Einschränkungen: Name und Parameterliste müssen übereinstimmen. Zugriffsschutz darf gelockert werden, aber nicht eingeschränkt. Ergebnistyp: Statt der Basisklasse darf die abgeleitete Klasse verwendet werden. Für den Methodenrumpf gelten keine Einschränkungen.

- **Was bedeutet der Zugriffsschutz-Modifier protected?**

Zugriff nur für abgeleitete Klassen (und die Klasse selbst) erlaubt

- **Kann eine Klasse in Java von mehreren Basisklassen abgeleitet sein? Kann eine Klasse in java mehrere Interfaces Implementieren?**

Nur eine Basisklasse; mehrere Interfaces

- **Nennen Sie eine Methode, die immer geerbt ist?**

toString() equals(Object x)

- **Was hat es mit der Klasse Object auf sich?**

Eine Klasse die von keiner abgeleitet ist erbt automatisch von der Wurzelklasse Objekt

- **Wie kann ein Konstruktor einer Basisklasse in der abgeleiteten Klasse verwendet werden?**

Durch super();
(Damit die geerbten Datenelemente initialisiert werden, muss in jedem Konstruktor einer abgeleiteten Klasse immer ein Basisklassen-Konstruktor aufgerufen werden
  super nur in einem abgeleiteten Konstruktor
  super nur einmal, als erste Anweisung im Rumpf
Falls kein expliziter Aufruf von super, Default Konstruktor der basisklasse
unbedingt erforderlich wenn Custom-Konstruktor für Basisklasse benötigt wird)

- **Wie können Programmabstürze verhindert werden?**

Durch Exceptions

- **Was können Ausnahmezustände sein?**

Unsinnige Eingaben (String statt int)
unvorhergesehene Zustände (Nenner=0)
Nullstelle soll innerhalb eines Intervalls berechnet werden, es gibt sie aber nicht…

- **Wie kann auf einen Ausnahmezustand im Programm reagiert werden?**

Eine Exception wird geworfen, später aufgefangen und ausgewertet.

- **Wie kann ein Fehlerzustand an die aufrufende Einheit einer Methode weitergeleitet werden?**

Durch das Werfen einer Exception die später aufgefangen wird.

- **Was versteht man unter Auffangen einer Exception?**

Nach dem try-Block mit dem regulären Code stehen die catch-Blöck(e) in denen die Exception aufgefangen wird und die Fehlerbehandlung durchgeführt wird.

- **Was passiert, wenn im try-Block eines Programms eine Exception auftritt?**

Unterbricht sofort den laufenden Code. Es wird im catch-Block weitergemacht (Fehlerbehandlung)

- **Was passiert, wenn eine Exception in einer Methode nicht behandelt wird?**

Nicht aufgefangene Exceptions werden an den nächsten übergeordneten Aufrufer weitergereicht.

- **Wie kann ein Programmierer erkennen, ob in einer Funktion Exceptions auftreten können?**

Durch Exceptionsignatur nach dem Methodenkopf
(returntyp methodenname (parameterliste) throws exceptiontyp1…)
und Dokumentation @throw exceptiontyp Text

- **Was versteht man unter einer Exception-Signatur?**

Wird nach dem Methodenkpf angegeben:
returntyp methodenname (parameterliste) throws exceptiontyp1…
zum Erkennen des Anwenders mit welcher Exception er rechnen muss z.B.: für catch blöcke

- **Was versteht man unter einer Methoden-Signatur?**

Methodenname + Übergabeparameter (methodenname(Parameter))

- **Was ist der Unterschied zwischen überladen und überschreiben (redefinieren) einer Methode?**

Überladene Methoden: mehrere Methoden mit gleichen Namen aber unterschiedlichen Parametern

Redefinieren: neue implementierung einer geerbten Methode, Einschränkungen:
Name und Parameterliste müssen übereinstimmen.
Zugriffsschutz darf gelockert werden, aber nicht eingeschränkt.
Ergebnistyp: Statt der Basisklasse darf die abgeleitete Klasse verwendet werden.
Für den Methodenrumpf gelten keine Einschränkungen.

- **Was bedeuten die reservierten Wörter final und finally in Java?**

Final: schlussendliche Wert einer Variable, kann nicht mehr verändert werden
finally: Block, nach den try- und catchBlöcken, der am Ende des Programms noch immer ausgeführt wird, egal ob Exception oder nicht.

- **Ist es in Java möglich eigene Exceptions zu definieren?**

Ja,

```
class Exceptionname extends Exception {
   Exceptionname () {  }
   Exceptionname (String message) {  super(message) ;  }
}
```

- **Was versteht man unter Exception-Chaining?**

Methode fängt Exception auf und gibt stattdessen eine andere Exception weiter. (so kommt es nicht zu ewig langen Exception-Signaturen)

- **Was unterscheidet eine Assertion von einer Exception?**

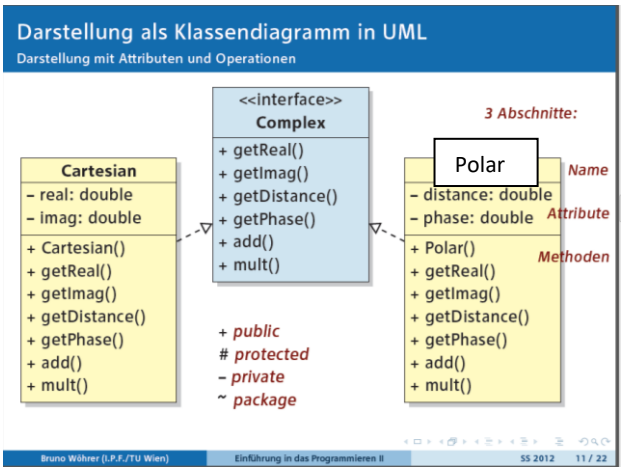
Assertion: Bedingung die geprüft wird (boolean==true), kann ausgestellt werden
Exception: Behandlung von Laufzeitfehler

- **Können Assertions in einem fertigen Programm aktiviert werden?**

Durch mitgeben vor –ea beim Start der JVM (-ea:classname für nur eine Klasse)

Zusätzliches:
Klasseninvarianten: Bedingungen die während der Laufzeit gelten müssen (Nenner!=0)
Preconditions: Randbedingungen, für die korrektes Ergebnis gilt

- **Zeichnen Sie ein einfaches Beispiel für ein Klassendiagramm in UML auf.**



- **Wie wird eine Vererbungsbeziehung im Klassendiagramm dargestellt?**

Cartesian erbt von Complex, Polar erbt von Complex

- **Wie kann man in einem Klassendiagramm den Zugriffsschutz erkennen?**

+ # - ~

- **Erklären Sie direkte und indirekte Rekursion.**

Direkte Rekursion: Methode ruft sich selbst immer wieder auf
indirekte Rekursion: Methoden rufen sich wechselseitig immer wieder auf

- **Was ist der entscheidende Punkt bei einer rekursiven Methode, damit es nicht zu einer unendlichen Aufruffolge kommt?**

Irgendwann soll Problem so klein sein, dass es ohne Rekursion lösbar ist

- **Kann es in einem funktionierenden Programm zu einem Stack-Überlauf kommen?**

Wichtig ist, dass irgendwann das Problem so klein sein muss, dass es ohne weitere Rekursion lösbar ist.

- **Was wird auf dem Programm-Stack abgelegt?**

Parameter und lokale Variablen (Last in, First out)

- **Nach welchen Kriterien kann die Komplexitat eines Programms beurteilt werden?**

Zeitkomplexität
Speicherkomplexität

- **Was sind typische Wachstumsordnungen bei Programmen?**

Konstant, logarithmisch, linear, überproportional (N*logN), quadratisch, kubisch
logarithmisch

- **Sie sehen ein Programm mit 4 verschachtelten Schleifen. Von welchem Wachstum können sie dann typischerweise ausgehen?**

N^4

- **Was ist das Problem bei exponetiell wachsender Laufzeit?**

Schon nach kurzer Zeit werden Probleme so groß, dass sie ewig zum Lösen benötigen.

- **Beschreiben Sie verbal, wie die binare Suche funktioniert.**

Voraussetzung: alle Elemente Sortiert
Idee:
Vergleiche den gesuchten Wert mit dem Element in der Mitte. Wenn Übereinstimung, dann sind wir fertig (gefunden), wenn gesuchter Wert kleiner, dann in vorderem Teil weitersuchen, wenn gesuchter Wert größer, dann im hinteren Teil weitersuchen. Wiederhole rekursiv, bis gefunden oder keine Elemente mehr vorhanden nicht gefunden.

Programme:

- **Binominalkoeffizienten:**

```
class Binom {
  static int binom (int a, int b) throws Exception {
    if (b>a) {
      return 0;
    }
    if(a<0||b<0){
      throw new Exception ("keine negativen werte");
    }
    if (a==b||b==0) {
      return 1;
    }
    else {
      int x = binom(a-1,b);
      int y = binom(a-1,b-1);
      return x+y;
    }
  }
}
```

- **Laufzeit**

```
public void test(double a[][]){
  int N=a.length;                    1
  for (int i=0; i<N; i++)            x+1
  for (int j=i; j<N; j++)            x*x/2+1
  if (i==j) a[i][j]=1;               x
  else a[i][j]=i+j;                  x*x/2-x
}                                    ➔ x^2+2x+2
```

Für Auswertung ist nur 1. Array interessant.

a[1][100] = 5
a[2][100] = 10
a[3][100] = 17
a[4][100] = 26

- **Exceptions**

```
class Except{
  public static int test(int a, int b){
    if (a<b) return b/a;
    else return a/b;
  }
  public static void main(String args[]){
    try {
      int z1 = Integer.parseInt(args[0]);
      int z2 = Integer.parseInt(args[1]);
      if(args.length>2){
        throw new ArrayIndexOutOfBoundsException();
      }
      System.out.println(test(z1,z2));
    }

     catch(ArithmeticException e) {
      System.out.println("keiner der Werte darf 0 sein");
    } catch(NumberFormatException e) {
      System.out.println("Bitte nur ganze Zahlen als Parameter eingeben");
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Bitte genau 2 Parameter eingeben");
    }
  }
}
```