Redundanz

- Anzahl der Bits $\geq H$ (H ist Entropie)
- Redundanz = Anzahl der Bits $- H$
- bei Redundanz Information vielleicht mehrfach
- Redundanz zur Fehlererkennung und -korrektur nutzbar
- Redundanz in Programmen kann Lesbarkeit verbessern
- Redundanz kann zu Inkonsistenzen führen
  (Information mehrfach, aber widersprüchlich)
- Komprimierung = Vermeidung von Redundanz

Information und Wissen

- Informationstheorie nutzt bekannte Wahrscheinlichkeiten
- Wahrscheinlichkeit hängt oft nur von Erwartung des Betrachters ab (wirklich alle Zahlen gleich wahrscheinlich?)
- Erwartung hängt vom Wissen des Betrachters ab
- Wissen über Darstellungsform/Bedeutung der Daten nötig
- ohne Wissen keine Wahrscheinlichkeit, keine Information
- Schrift bzw. Sprache entspricht allgemeinem Wissen über Darstellungsform von Daten bzw. Texten

Komprimierung

- Informationsgehalt weist auf Darstellungsmöglichkeit hin
  - jeder Wert w mit etwa $I(p_w)$ Bits optimal dargestellt
- Wahrscheinlichkeit/Informationsgehalt wissensabhängig
  - viel Wissen $\rightarrow$ bessere Komprimierung
- *verlustbehaftete* Komprimierungen (Audio, Bilder, Video) entfernen unwichtige Details (z.B. Rauschen) um vorhandenes Wissen besser Nutzen zu können
- kaum nutzbares Wissen für verlustfreie Komprimierungen

Verschlüsselung und Entschlüsselung

- Daten ohne Wissen über Darstellungsform wertlos
- Verschlüsselung = Umformung von bekannter Darstellungsform auf unbekannte (mit derselben Information)
- Entschlüsselung = Umformung von unbekannter Darstellungsform auf bekannte (mit derselben Information)
- Während Ver-/Entschlüsselung muss auch die unbekannte Darstellungsform bekannt sein (Schlüssel)
- getrennte Schlüssel für Ver-/Entschlüsselung möglich
- viel Redundanz $\Rightarrow$ Schlüssel leichter „knackbar“
  (Kind erlernt Sprache aus Wiederholungen)

Verstecken von Information

- Wissen über Datendarstellung $\rightarrow$ Information
- unterschiedliches Wissen bei selben Daten
  $\rightarrow$ unterschiedliche Informationen
- verstecken von Information in scheinbar harmlosen Daten
  „Morgen wird es regnen“ heißt „Anschlag erfolgt morgen“
- moderne Variante: Verstecken digitaler Daten z.B. im Rauschen von Bildern, Filmen, etc.
- geheime Daten und Schlüssel (Bilder, Filme, etc.) kaum als solche erkennbar

Versteckte Information in Programmen

- (schädlicher) Code auch in Programmen versteckbar
- z.B. Abfrage von Hotkeys, Weiterleitung geheimer Daten
- durch geschickte Namensgebung und verkomplizierten Code auch durch Code Reviews kaum auffindbar
- „The International Obfuscated C Code Contest“
- absichtliches versus unabsichtliches Verstecken
- zweifelhafter, unverständlicher Code immer schlecht (egal ob absichtlich oder nicht)

Pakete in Java

Pakete

- Paket = alle Klassen und Interfaces in selbem Verzeichnis
- Paket-Sichtbarkeit (als Default): im ganzen Paket sichtbar
- *Default-Paket:* das Verzeichnis, in dem man sich befindet (meist das, in dem der Java-Interpreter gestartet wird)
- Paket als *Namensraum:* nur Klassen und Interfaces direkt sichtbar, die im Default-Paket stehen (oder zum vordefinierten Paket `java.lang` gehören, z.B. `Object`, `String`, ...)
- Klassen und Interfaces aus anderen Paketen *qualifiziert* zugreifbar (z.B. `java.util.ArrayList<X>`, `java.io.FileReader`)

Qualifizierte Namen

- `import`-Anweisungen (am Anfang von `.java`-Dateien) machen Namen aus anderen Paketen direkt zugreifbar:
  - `import java.util.ArrayList;` (erlaubt `ArrayList<X>`)
  - `import java.util.*;` (für alles aus `java.util`)
- ohne `import`-Anweisung qualifizierter Name notwendig: überall `java.util.ArrayList<X>` als Typ, gleiche Bedeutung
- was mit `java` beginnt ist Java-Standard-Klasse / -Interface
- sonst: Pfadname (im Dateisystem) relativ zu *Classpath*
- *Classpath:* Liste von Verzeichnissen (Option `-cp` von `java`)

Paket-Deklaration

- für qualifizierten Zugriff `package`-Anweisung nötig (erste Anweisung in `.java`-Datei), z.B.: `package mypack.MyClass;`
- Klasse/Interface heißt `MyClass` und steht in `MyClass.java`
- `MyClass.java` muss im Verzeichnis `mypack` enthalten sein
- `mypack` muss im Default-Paket enthalten sein
- Ausführung (aus Default-Paket) durch `java mypack.MyClass`
- tiefe Schachtelung möglich: `pack1.pack2.mypack.MyClass`

Java-Archive

- `.jar`-Datei enthält `.class`-Dateien (aus `.java`-Dateien erzeugt) in komprimierter Form, meist ganzes Paket
- Erzeugung z.B. durch:

  ```
  jar cfe my.jar mypack.MyClass mypack/*.class
  ```

- Ausführung: `java -jar my.jar`
- fertige Java-Programme meist als `.jar`-Dateien geliefert
- `.jar`-Dateien auch wie Pakete verwendbar (Classpath)

Praktischer Umgang mit Paketen

- Umgang mit Paketen und Archiven schwierig
- zahlt sich für kleine Programme kaum aus
- große Programme in logisch möglischst klar voneinander getrennte Pakete zerlegen, jedes Paket mehrere Klassen
- „Programmieren im Großen“
- Organisation spiegelt Softwarearchitektur wider
- gute Architektur erfordert viel Erfahrung