Aufgabe 2:

Vor- und Nachteile der Gleichbenennung einer Java-Datei zu ihrer beinhaltenden Klasse:

(Musste ich googlen, weil ich es im Buch entweder überlesen habe oder es nicht näher behandelt wurde): https://forums.spoken-tutorial.org/question/781/file-name-matching-class-name/

Nachteil: Ist die Klasseöffentlich, könnte es beim Kompilieren zu Problemen kommen, insbesondere wenn mehrere Klassen existieren, die dann Namenskonflikte auslösen.

Vorteil: Werden nur private Inner-Classes benötigt, die von außen nicht gebraucht werden, dann kann die Datei nach ihrer Hauptklasse benannt werden. Der Compiler aber wird aus allen gefundenen Klassen Dateien erzeugen. Besser eine Datei pro Klasse.

Aufgabe 4:

Zuerst einmal ist der Klassenname „DoEverything“ nicht aussagekräftig und besagt nichts darüber, was die Klasse tut oder welche Objekte sie initialisieren lassen könnte.

Zum Zweiten ist die int-Variable INTERESTRATE entweder eine Konstante und es wurde das Keyword final davor vergessen oder es ist keine und dann ist die Großschreibung nicht den Konventionen entsprechend.

Ansonsten ist nicht ersichtlich in welchem Zusammenhang die klassenglobalen Variablen (int INTERESTRATE und String defaultFilePath) stehen und vielleicht sogar beide final (und uppercase) sein sollten.

Zuletzt geht aus den Angaben nicht hervor, ob die Methoden der Klasse in einem logischen und logistischen Zusammenhang stehen oder unnötigerweise zusammen in eine Klasse geschrieben wurden. Das eine sind Berechnungen, das andere Datei-Operationen. Werden keine Ergebnisse in Dateien gespeichert, ist eines davon unnötig.

Aufgabe 5:

„Encapsulation“ nennt man auf Deutsch auch Datenkapselung. Hierbei ist gemeint, dass Klassen, sowie ihre Attribute und Methoden mithilfe von Modifiern in ihrer Sichtbarkeit und somit ihrem zugriff eingeschränkt werden können. In erster Linie soll dies also daten vor ungewollter Manipulation schützen. Dabei lassen sich mit Settern und Gettern kontrollierte und bedingte Zugriffe ermöglichen. Funktionalitäten werden hier nur indirekt eingeschränkt, indem man Klassen und Methoden private setzt, falls nur intern benötigt.

Aufgabe 1:

Verwendung: java [Optionen] <Hauptklasse> [args...]
(zur Ausführung einer Klasse)
oder java [Optionen] -jar <JAR-Datei> [args...]
(zur Ausführung einer JAR-Datei)
oder java [Optionen] -m <Modul>[/<Hauptklasse>] [args...]
java [Optionen] --module <Modul>[/<Hauptklasse>] [args...]
(zur Ausführung der Hauptklasse in einem Modul)
oder java [Optionen] <Quelldatei> [args]
(zur Ausführung eines Programms mit einer Quelldatei)

Argumente, die auf die Hauptklasse, die Quelldatei, -jar <JAR-Datei>, -m oder --module <Modul>/<Hauptklasse> folgen, werden als Argumente für die Hauptklasse übergeben.

Dabei umfasst "Optionen" Folgendes:

-cp <Klassensuchpfad mit Verzeichnissen und ZIP-/JAR-Dateien>
-classpath <Klassensuchpfad mit Verzeichnissen und ZIP-/JAR-Dateien>
--class-path <Klassensuchpfad mit Verzeichnissen und ZIP-/JAR-Dateien>
Eine durch ; getrennte Liste mit Verzeichnissen, JAR-Archiven
und ZIP-Archiven, in denen nach Klassendateien gesucht wird.
-p <Modulpfad>
--module-path <Modulpfad>...
Eine durch ; getrennte Liste mit Elementen, von denen jedes Element ein Dateipfad
zu einem Modul oder einem Verzeichnis mit Modulen ist. Jedes Modul ist entweder
ein modulares JAR oder ein entpacktes Modulverzeichnis.
--upgrade-module-path <Modulpfad>...
Eine durch ; getrennte Liste mit Elementen, von denen jedes Element ein Dateipfad
zu einem Modul oder einem Verzeichnis mit Modulen ist,
um upgradefähige Module im Laufzeitimage zu ersetzen. Jedes Modul ist entweder
ein modulares JAR oder ein entpacktes Modulverzeichnis.
--add-modules <Modulname>[,<Modulname>...]
Root-Module, die zusätzlich zum anfänglichen Modul aufgelöst werden sollen.
<Modulname> kann auch wie folgt lauten: ALL-DEFAULT, ALL-SYSTEM,
ALL-MODULE-PATH.
--enable-native-access <Modulname>[,<Modulname>...]
Damit kann der Code in Modulen auf Code und Daten außerhalb der JRE zugreifen.
<Modulname> kann auch ALL-UNNAMED sein, um den Code im Classpath anzugeben.
--list-modules
Listet beobachtbare Module auf und beendet den Vorgang
-d <Modulname>
--describe-module <Modulname>
Beschreibt ein Modul und beendet den Vorgang
--dry-run Erstellt eine VM und lädt die Hauptklasse, führt aber nicht die Hauptmethode aus.
Die Option "--dry-run" kann nützlich sein, um die
Befehlszeilenoptionen, wie die Modulsystemkonfiguration, zu validieren.
--validate-modules
Validiert alle Module und beendet den Vorgang
Die Option "--validate-modules" kann nützlich sein, um

Konflikte und andere Fehler mit Modulen auf dem Modulpfad zu ermitteln.
-D<Name>=<Wert>
Legt eine Systemeigenschaft fest
-verbose:[class|module|gc|jni]
Aktiviert die Verbose-Ausgabe für das angegebene Subsystem
-version Gibt die Produktversion an den Fehlerstream aus und beendet den Vorgang
--version Gibt die Produktversion an den Outputstream aus und beendet den Vorgang
-showversion Gibt die Produktversion an den Fehlerstream aus und setzt den Vorgang fort
--show-version
Gibt die Produktversion an den Outputstream aus und setzt den Vorgang fort
--show-module-resolution
Zeigt die Modulauflösungsausgabe beim Start an
-? -h -help
Gibt diese Hilfemeldung an den Fehlerstream aus
--help Gibt diese Hilfemeldung an den Outputstream aus
-X Gibt Hilfe zu zusätzlichen Optionen an den Fehlerstream aus
--help-extra Gibt Hilfe zu zusätzlichen Optionen an den Outputstream aus
-ea[:<packagename>...|:<classname>]
-enableassertions[:<packagename>...|:<classname>]
Aktiviert Assertions mit angegebener Granularität
-da[:<packagename>...|:<classname>]
-disableassertions[:<packagename>...|:<classname>]
Deaktiviert Assertions mit angegebener Granularität
-esa | -enablesystemassertions
Aktiviert System-Assertions
-dsa | -disablesystemassertions
Deaktiviert System-Assertions
-agentlib:<libname>[=<options>]
Lädt die native Agent Library <libname>. Beispiel: -agentlib:jdwp
siehe auch -agentlib:jdwp=help
-agentpath:<pathname>[=<options>]
Lädt die native Agent Library mit dem vollständigen Pfadnamen
-javaagent:<jarpath>[=<options>]
Lädt den Java-Programmiersprachen-Agent, siehe java.lang.instrument
-splash:<imagepath>
Zeigt den Startbildschirm mit einem angegebenen Bild an
Skalierte HiDPI-Bilder werden automatisch unterstützt und verwendet,
falls verfügbar. Der nicht skalierte Bilddateiname (Beispiel: image.ext)
muss immer als Argument an die Option "-splash" übergeben werden.
Das am besten geeignete angegebene skalierte Bild wird
automatisch ausgewählt.
Weitere Informationen finden Sie in der Dokumentation zur SplashScreen-API
@argument files
Eine oder mehrere Argumentdateien mit Optionen
--disable-@files
Verhindert die weitere Erweiterung von Argumentdateien
--enable-preview
Lässt zu, das Klassen von Vorschaufeatures dieses Release abhängig sind
Um ein Argument für eine lange Option anzugeben, können Sie --<Name>=<Wert> oder
--<Name> <Wert> verwenden.

Aufgabe 3:

Um ersichtlich zu machen, dass es der Hauptpunkt des Programmes ist, sollte die Klasse entweder direkt Main heißen oder im Zusammenhang mit dem Projekt einen so aussagekräftigen Namen bekommen, der im Kontext zu anderen Klassen ersichtlich werden lässt, dass diese Klasse den zentralen Punkt des Projektes darstellt / beinhaltet.

Aufgabe 6:

Wie in Aufgabe 5 bereits erwähnt, werden Variablen und Methoden einer Klasse / Instanz verborgen, um ungewollte bzw. unbefugte Änderungen zu vermeiden. Durch die getter- und setter-Methoden wird ein kontrollierter Zugriff sichergestellt, der bestimmte Fälle und Szenarien abfangen kann, die die Änderung von Variablen betreffen. Vor allem können hier Abhängigkeiten relativiert werden, anstatt an jeder Stelle hart zu coden. So kann eine Methode Werte in Bezug auf ihre Variablen liefern und wird diese dann verändert, muss dies nur einmal geschehen und nicht an allen Stellen, wo sie aufgerufen wird. Dies erleichtert somit auch die Wartbarkeit ungemein. Denn so bestehen keine redundanten oder bei Änderung dann inkonsistenten Codeabschnitte.

Aufgabe 7:

Die Vererbung haben wir als Möglichkeit der Wiederverwendung von Code kennengelernt. Hierbei werden abstrahierbare Eigenschaften und Verhaltensweisen in eine übergeordnete (und ggf. auch abstrakte) Klasse geschrieben, so dass alle davon erbenden Klassen trotz ihrer Abweichungen (Polymorphismus) die Gemeinsamkeiten nutzen können und diese somit nicht nochmal beschrieben werden müssen.

Bezüglich der anderen Angabe Aggregation habe ich soeben folgendes gelesen:

Aggregation (allgemein Anhäufung/Ansammlung) ist die Integration bzw. Existenz von Objekten ineinander. Ein Sonderfall davon ist die Komposition (Zusammenstellung), die eine zwingende Notwendigkeit von Objekten ergibt. So kann zum Beispiel ein Baseballspiel nicht ohne Schläger, Ball und Spieler existieren. Hierbei können auch Anzahlen bzw. Intervalle angegeben werden. Dies hat also nichts mit der Frage zu tun. Maximal indirekt, da nicht jeder weitere Ball/Schläger/Spieler neu erstellt werden muss, sondern nur einmal definiert wird, wie diese aufgebaut sind und wie sie funktionieren.

(https://www.learnj.de/doku.php?id=diagrams:diagrams:aggregation:start)

(https://www.google.com/search?q=aggregation+java&rlz=1C1CHBF_enDE1126DE1126&oq=aggregation&gs_lcrp=EgZjaHJvbWUqDAgBEAAYFBiHAhiABDITCAAQRRgnGDsYRhj5ARiABBiKBTIMCAEQABgUGIcCGIAEMgcIAhAAGIAEMgcIAxAAGIAEMgcIBBAAGIAEMgcIBRAAGIAEMgcIBhAAGIAEMgcIBxAAGIAEMgcICBAAGIAEMgcICRAAGIAE0gEIMzc0M2owajmoAgCwAgE&sourceid=chrome&ie=UTF-8#:~:text=Komposition.%20Die%20Komposition,abh%C3%A4ngig%20sind.)