|  |
| --- |
| Högskolan i Gävle |
| Laborationsrapport |
| Objektorienterad Design och Programmering, 7,5 hp  HT 19, period 1  Laboration 2  av  *Thomas Lundgren*  *900802-1835*  [*thomaslundgren@live.com*](mailto:thomaslundgren@live.com) |
|  |
| **Institutionen för matematik, natur- och datavetenskap**  **Högskolan i Gävle**  S-801 76 Gävle, Sweden |
|  |
|  |

**Innehållsförteckning**

[1 Inledning 1](#_Toc19863073)

[1.1 Syfte och riktlinjer 1](#_Toc19863074)

[1.2 Specifikation av räknare 1](#_Toc19863075)

[2 Metod och utförande 2](#_Toc19863076)

[2.1 Gränssnittet Counter och de abstrakta klasserna AbstractCounter och LinkedAbstractCounter 2](#_Toc19863077)

[2.2 Klasserna Counter60, Counter24 och DecreasingCounter10 2](#_Toc19863078)

[2.3 Klassen ClockCounter 2](#_Toc19863079)

[2.4 Enhetstester med JUnit 3](#_Toc19863080)

[2.5 Klassdiagram 3](#_Toc19863081)

[3 Resultat 4](#_Toc19863082)

[3.1 Räknaren 4](#_Toc19863083)

[3.2 Klassdiagram 4](#_Toc19863084)

[3.3 Enhetstester med JUnit 5](#_Toc19863085)

[3.4 Riktlinjer och designprinciper 5](#_Toc19863086)

[4 Diskussion 6](#_Toc19863087)

[4.1 Implementation av AbstractCounter och LinkedAbstractCounter 6](#_Toc19863088)

[4.2 Enhetstester med JUnit 6](#_Toc19863089)

[5 Referenser 7](#_Toc19863090)

1. Inledning
   1. Syfte och riktlinjer

I denna laboration skulle interface och klasser designas och implementeras som i slutänden skulle utgöra delarna till en alarmklocka. En del kod var given. Laborationen bygger vidare på laboration 1 och de interface och klasser som skapades i det arbetet. Syftet med denna laboration var (utöver det syfte som angavs i laboration 1) att studenten skulle:

* förstå och använda begrepp såsom: paket (package), aggregation, komposition, delegering, utökat beteende genom arv (eller komposition/delegering)
* toString()
* manipulera strängar
* prova på att använda olika typer av collections, t.ex. List, LinkedList, Map, HashMap, samt iterator-begreppet
* prova på att använda klasser för mönstermatchning med reguljära uttryck
* sätta sig in i och förstå given kod
* prova att slänga fel (throw RuntimeException)
* förstår och använda klassdiagram i UML och träna på att identifiera/analysera relationer mellan klasser
* använda tester (JUnit) och därigenom avlusa programvara och säkerställa robust kod

Generella riktlinjer och principer som skulle tas i beaktande och i största möjliga mån följas var:

* Information hiding
* Program to an interface
* Använd högsta möjliga abstraktionsnivå för objektreferenser
* Single Responsibility Principle
* Objekt ska vara ”robusta”, dvs att lämplig felhantering används och att logiken är utformad på ett tillförlitligt sätt. Detta säkerställs med hjälp av enhetstester.
  1. Specifikation av alarmklocka

Laborationen skulle resultera i en klass som beskriver en alarmklocka. Denna skulle använda sig av den kod som skrevs i laboration 1.

Alarmklockan skulle kunna ha flera alarm som ställs och aktiveras vid olika tidpunkter.

Koden skulle testas med JUnit-tester (enhetstester).

1. Metod och utförande
   1. Utökat beteende hos Counter-klasserna

I laboration 1 skapades interfaces och klasser som specificerade och implementerade räknare. Dessa behövde utökas så att räknaren kunde ställas till en specifik tid.

Interfacet SettableCounterType skapades. Detta fick ärva specifikationen i CounterType genom att utöka (extend) interfacet. Metoden setCount(int value) lades till.

En ”ställbar” version av klassen AbstractCounter skapades. Denna döptes till SettableCounter. Denna klass utökar AbstractCounter och implementerar interfacet SettableCounterType. Således måste klassen implementera metoden setCount(int value) samt alla metoder som definieras i AbstractCounter.

Då klassen LinkedAbstractCounter hade skapats i laboration 1 skapades en ställbar, utökad version av den. Denna döptes till SettableLinkedCounter. Denna klass utökar AbstractLinkedCounter och implementerar interfacet SettableCounterType.

Klasserna Counter24 och Counter60 ändrades så att de nu istället utökar SettableLinkedCounter. De är alltså räknare som har en inre räknare och som kan ställas. Klassen Counter7, som skulle användas för att räkna veckodagar, skapades och fick utöka klassen SettableCounter.

* 1. Gränssnittet TimeType och klassen Time

Gränssnittet TimeType och klassen Time skapades för att representera tid i alarmklockan som skulle skapas. All kod för dessa var given. Klassen Time utökades något med felhantering. Vid en tidpunkt i utvecklingen fick klassen Time implementera interfacet Comparable<Time> för att möjliggöra jämförelser av instanser av klassen. Senare i utvecklingen gjordes upptäckten att detta inte var nödvändigt, men funktionaliteten fick kvarstå.

* 1. Gränssnittet AlarmType och klassen Alarm

Metodsignaturerna i gränssnittet AlarmType var given. Gränssnittet specificerar funktionaliteten hos ett alarm. En liten del av koden för klassen Alarm var given och resterande kod implementerades. Även här lades felhantering till.

* 1. Klassen AlarmManager

Klassen AlarmManager är den del av alarmklockan som är ansvarig för att lägga till, ta bort och handha alarm. Den har också som uppgift att kontrollera om det är dags att aktivera något alarm när klockans tid avancerar. Koden för klassen var given.

* 1. Gränssnittet AlarmActionType och klassen PrintAlarmAction

Gränssnittet AlarmActionType innehåller endast två metoder, alarmActivated() och alarmDeactivated(). Dessa metoder tillhandahåller de aktioner, de *funktioner* som ska utföras då ett alarm aktiveras. En implementation av gränssnittet, klassen PrintAlarmAction, skrevs.

* 1. Gränssnitten ClockType och AlarmClockType, klasserna WeekClock och WeekAlarmClock

Gränssnittet AlarmClockType var givet. I detta specificerades sju metoder som beskriver funktionaliteten hos en alarmklocka enligt nedan:

public interface AlarmClockType { public void tickTock(); public void setTime(TimeType time); public void addAlarm(AlarmType larm); public void removeAlarm(AlarmType alarm); public Collection<AlarmType> getAlarms(); public TimeType getTime(); public String toString(); }

Metoderna tickTock, setTime(), getTime() och toString() bröts ut till ett mer generellt interface kallat ClockType och tre metoder lades till: startClock(), stopClock() och resetClock(). ClockType skapades för att erbjuda en mer generell implementation av en klocka. Gränssnittet AlarmClockType fick sedan utöka ClockType och specificera metoderna addAlarm(), removeAlarm() och getAlarms().

* 1. UML-klassdiagram

Ett UML-klassdiagram skapades för varje paket/modul med ritverktyget draw.io.

* 1. Enhetstester med JUnit

Enhetstester skrevs för varje konkret implementation (klass). Enhetstesterna skrevs för att största möjliga mån täcka in gränsvärden och för att säkerställa att felhanteringen fungerar som den ska.

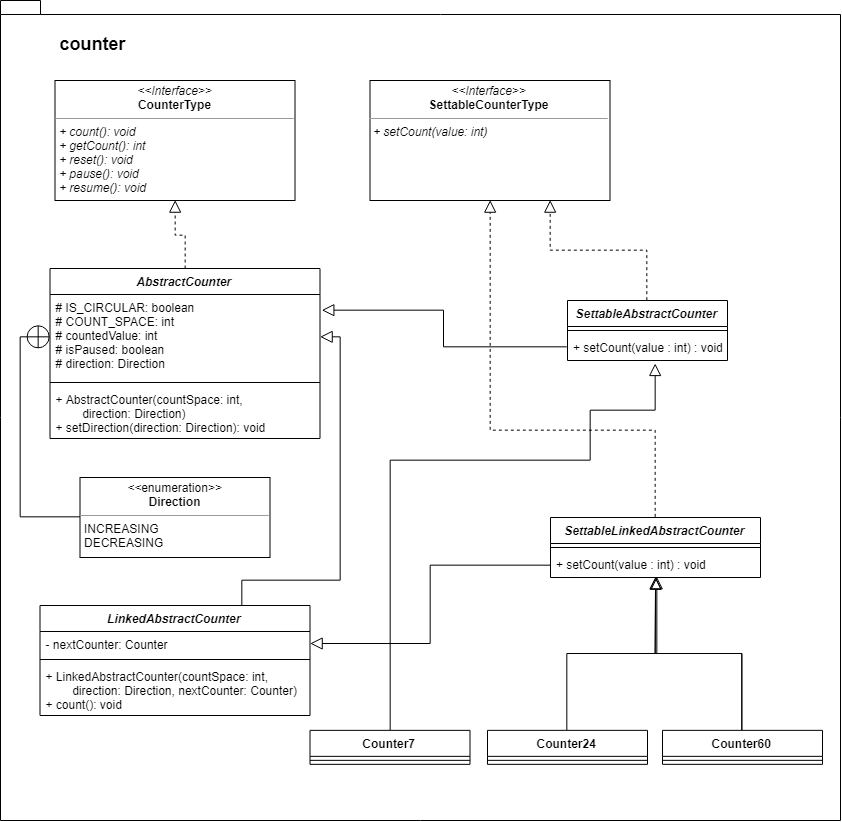
Det i Eclipse inbyggda plug-in-programmet EclEmma användes för att avgöra hur stor del av koden som täcks av enhetstesterna.

1. Resultat

Resultaten av laborationen redovisas här under rubriker som motsvarar de fyra paket/moduler som skapats, nämligen counter, time, alarm och clock. Efter en beskrivning av de fyra modulerna följer en mer översiktlig genomgång av laborationens resultat.

För källkod, se *6 - Bilaga A – källkod*.

* 1. Counter

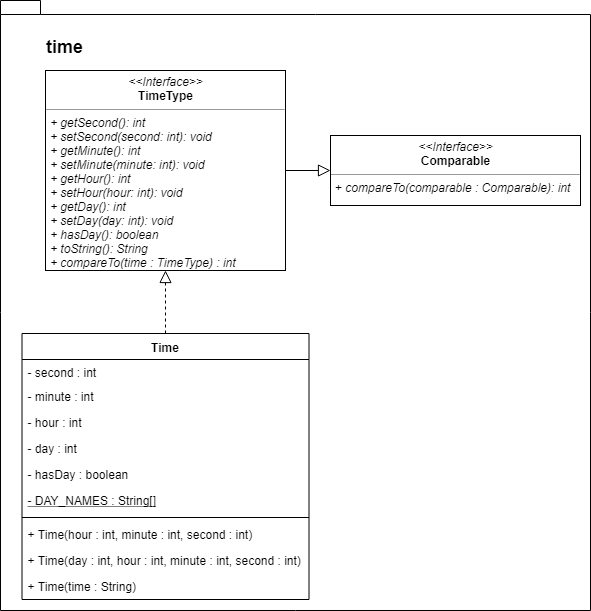
Figur 1 visar alla gränssnitt och klasser i paketet counter.

Figur 1: Klassdiagram för alla gränssnitt och klasser i paketet counter.

All funktionalitet som specificeras i de abstrakta datatyperna utökas och implementeras genom arv och implementation av gränssnitt.

* 1. Time

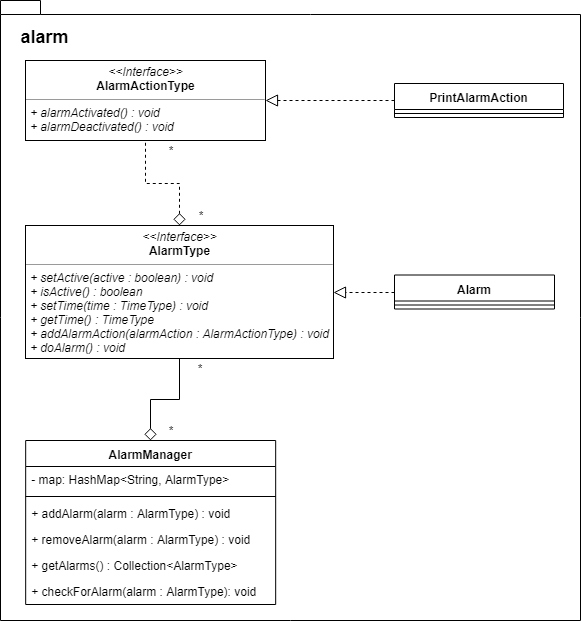
Figur 2 gränssnittet TimeType och klassen Time som utgör time-paketet.



Figur 2: Klassdiagram för paketet time.

* 1. Alarm

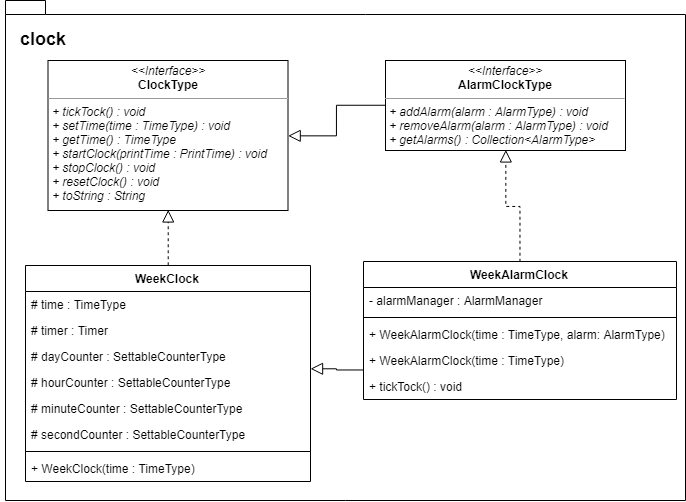
Figur 3 visar klassdiagrammet för paketet alarm. Gränssnittet AlarmType är beroende av gränssnittet TimeType i paketet time. Detta visas inte i klassdiagrammet för att hålla det överskådligt. För en beskrivning av beroenden mellan klasser i olika paket, se *3.5 - Beroenden mellan klasser*.

* 1. Clock

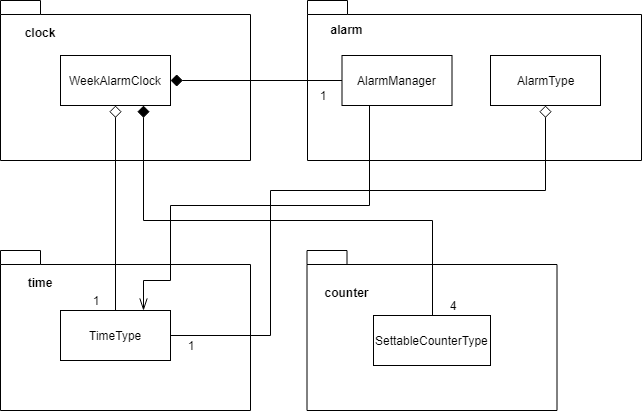
Figur 3: Klassdiagram för paketet alarm.

Figur 4 visar klassdiagrammet för paketet clock. Klassen WeekAlarmClock är det viktigaste resultatet i laborationen. Den erbjuder funktionaliteten som skulle uppnås. WeekAlarmClock har beroenden av klasser utanför paketet clock, nämligen AlarmManager i paketet alarm och SettableCounterType i paketet counter. WeekAlarmClock delegerar hanteringen av Alarm-objekt till AlarmManager.

För en beskrivning av beroenden mellan klasser i olika paket, se *3.5 - Beroenden mellan klasser*.

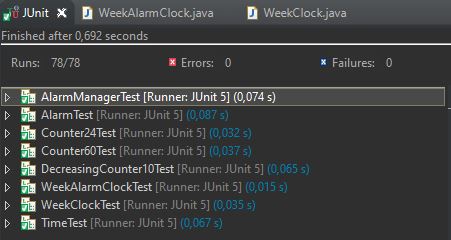
* 1. Beroenden mellan klasser

Figur 4: Klassdiagram för paketet clock.

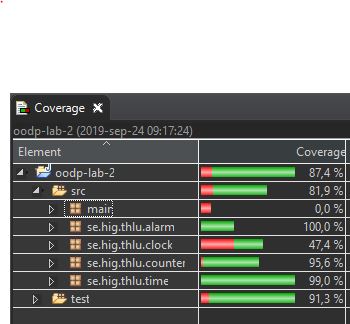
Figur 5 visar en något förenklad bild över de beroenden som förekommer över paketens gränser.

Figur 5: Beroenden över paketens gränser.

* 1. Enhetstester med JUnit

Enhetstester skrevs för alla klasser. Totalt skrevs 78 enhetstester. Figur 6 visar att alla enhetstester exekveras utan fel.

Figur 6: Resultatet av att köra alla JUnit-tester i Eclipse.

EclEmma användes för att mäta hur stor del av källkoden som täcks av enhetstesterna, se Figur 7. EclEmma visar att 81,9 % av källkoden testas i enhetstesterna. Paketet clock har sämst testtäckning. Detta beror på att metoderna startClock(), stopClock() och resetClock() inte testats.

Figur 7: Resultatet av exekvering av EclEmma.

* 1. Designprinciper och riktlinjer

Genom dessa designval har de riktlinjer och designprinciper som specificerats följts:

* Program to an interface: Gränssnitt/abstraktioner har skapats för alla abstrakta datatyper där det har ansetts fördelaktigt.
* Alla objektreferenser är av typen med allra högst abstraktionsnivå (gränssnitt/abstrakt klass).
* Information hiding har åstadkommits genom att alla instansvariabler har definierats med de mest restriktiva tillgångsmodifierarna möjliga.
* Koden är robust då inga null-referenser tillåts och enhetstester finns för en stor del av koden och många use-cases.
* Funktionalitet som utökar funktionaliteten i den givna koden har definierats i ett nytt Interface/en ny klass i enlighet med Open-Closed Principle.
* Varje Interface/klass har ett begränsat, sammanhängande ansvarsområde i enlighet med Single Responsibility Principle.
* Delegering har använts där det är passande. Ett exempel på detta är i klassen WeekAlarmClock där hanteringen av AlarmType-objekt delegeras till AlarmManager.

1. Diskussion
   1. Implementation av AbstractCounter och LinkedAbstractCounter

Funktionaliteten för att länka samman räknare extraherades från AbstractCounter och lades i LinkedAbstractCounter istället. Detta för att undvika Null-referenser i instansvariabeln nextCounter då användaren önskar skapa en fristående räknare. I min mening resulterar detta i mer modulär och robust kod. Klientkod tvingas vara mer explicit i vilken sorts räknare som önskas, men potentiella problem med Null-referenser elimineras.

* 1. Enhetstester med JUnit

I enhetstesterna används Roy Osheroves föreslagna namngivningskonvention. Den går ut på att ett enhetstests namn delas in i tre delar. Han beskriver det som:

[UnitOfWork\_StateUnderTest\_ExpectedBehavior]

vilket kan läsas på hans blogg [1].

1. Referenser
2. Bilaga A – källkod

Här redovisas all källkod som har producerats under laborationen. Kod ifrån föregående laboration redovisas inte här. Mindre ändringar kan ha förekommit i koden från föregående laboration.

* 1. Gränssnittet SettableCounterType

package se.hig.thlu.counter;  
  
public interface SettableCounterType extends CounterType {  
  
 public void setCount(int value);  
  
}

* 1. Klassen SettableAbstractCounter

a package se.hig.thlu.counter;  
  
abstract class SettableAbstractCounter extends AbstractCounter implements SettableCounterType {  
  
 public SettableAbstractCounter(int countSpace, *Direction* direction) {  
 super(countSpace, direction);  
 }  
  
 *@Override*  
 public void setCount(int value) {  
 if (value < 0 || value > COUNT\_SPACE) {  
 throw new IllegalArgumentException(  
 "Value cannot be less than zero or exceed the specified count space of the Counter!");  
 }  
 this.countedValue = value;  
 }  
  
}

* 1. Klassen SettableLinkedAbstractCounter

package se.hig.thlu.counter;  
  
import java.util.Objects;  
  
abstract class SettableLinkedAbstractCounter extends LinkedAbstractCounter implements SettableCounterType {  
  
 public SettableLinkedAbstractCounter(int countSpace, Direction direction, CounterType nextCounter) {  
 super(countSpace, direction, Objects.requireNonNull(nextCounter));  
 }  
  
 @Override  
 public void setCount(int value) {  
 if (value < 0 || value >= COUNT\_SPACE) {  
 throw new IllegalArgumentException("Value cannot be set to less than zero or more than the specified count space of the Counter!");  
 }  
 this.countedValue = value;  
 }  
}

* 1. Gränssnittet TimeType

package se.hig.thlu.time;  
  
public interface TimeType extends Comparable<TimeType> {  
 public int getSecond();  
 public void setSecond(int second);  
 public int getMinute();  
 public void setMinute(int minute);  
 public int getHour();  
 public void setHour(int hour);  
 public int getDay();  
 public void setDay(int day);  
 public boolean hasDay();  
 public String toString();  
}

* 1. Klassen Time

package se.hig.thlu.time;  
  
import java.util.regex.Matcher;  
import java.util.regex.Pattern;  
  
public class Time implements TimeType {  
  
 private int second, minute, hour, day;  
 private static final String[] ***DAY\_NAMES*** = { "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun" };  
 private boolean hasDay = false;  
  
 public Time(int hour, int minute, int second) {  
 setHour(hour);  
 setMinute(minute);  
 setSecond(second);  
 }  
  
 public Time(int day, int hour, int minute, int second) {  
 this(hour, minute, second);  
 setDay(day);  
 }  
  
 public Time(String time) {  
 // kolla format Hh:Mm:Ss (H: 0-2, h: 0-9, M: 0-5, m: 0-9, S: 0-5, s: 0- 9)  
 Pattern plainPattern = Pattern.*compile*("^(2[0-3]|[01]?[0-9]):([0-5]?[0- 9]):([0-5]?[0-9])$");  
 Matcher plainMatcher = plainPattern.matcher(time);  
 // kolla format Day Hh:Mm:Ss (Day: Mon, Tue, ..., H: 0-2, h: 0-9, M: 0-5, m:  
 // 0-9, S: 0-5, s: 0-9)  
 Pattern weekDayPattern = Pattern  
 .*compile*("^(Mon|Tue|Wed|Thu|Fri|Sat|Sun) (2[0-3]|[01]?[0- 9]):([0-5]?[0-9]):([0-5]?[0-9])$");  
 Matcher weekDayMatcher = weekDayPattern.matcher(time);  
 if (weekDayMatcher.find()) {  
 String[] parts = time.split(" |:");  
 for (int day = 0; day < ***DAY\_NAMES***.length; day++)  
 if (***DAY\_NAMES***[day].equals(parts[0]))  
 setDay(day);  
 setHour(Integer.*parseInt*(parts[1]));  
 setMinute(Integer.*parseInt*(parts[2]));  
 setSecond(Integer.*parseInt*(parts[3]));  
 } else if (plainMatcher.find()) {  
 String[] parts = time.split(":");  
 setHour(Integer.*parseInt*(parts[0]));  
 setMinute(Integer.*parseInt*(parts[1]));  
 setSecond(Integer.*parseInt*(parts[2]));  
 } else  
 throw new RuntimeException("Illegal time format: " + time);  
 }  
  
 *@Override*  
 public int getSecond() {  
 return second;  
 }  
  
 *@Override*  
 public void setSecond(int second) {  
 if (isNegative(second)) {  
 throw new IllegalArgumentException("Seconds cannot be less than zero!");  
 }  
 this.second = second % 60;  
 }  
  
 *@Override*  
 public int getMinute() {  
 return minute;  
 }  
  
 *@Override*  
 public void setMinute(int minute) {  
 if (isNegative(minute)) {  
 throw new IllegalArgumentException("Minutes cannot be less than zero!");  
 }  
 this.minute = minute % 60;  
 }  
  
 *@Override*  
 public int getHour() {  
 return hour;  
 }  
  
 *@Override*  
 public void setHour(int hour) {  
 if (isNegative(hour)) {  
 throw new IllegalArgumentException("Hours cannot be less than zero!");  
 }  
 this.hour = hour % 24;  
 }  
  
 *@Override*  
 public boolean hasDay() {  
 return hasDay;  
 }  
  
 *@Override*  
 public int getDay() {  
 return day;  
 }  
  
 *@Override*  
 public void setDay(int day) {  
 if (isNegative(day)) {  
 throw new IllegalArgumentException("Day cannot be less than zero!");  
 }  
 this.day = day % 7;  
 hasDay = true;  
 }  
  
 *@Override*  
 public int compareTo(TimeType time) {  
 int thisSeconds = 0;  
 int compareToSeconds = 0;  
 thisSeconds = second + minute \* 60 + hour \* (60 \* 60) + day \* (24 \* 60 \* 60);  
 compareToSeconds = time.getSecond() + time.getMinute() \* 60 + time.getHour() \* (60 \* 60)  
 + time.getDay() \* (24 \* 60 \* 60);  
  
 if (thisSeconds > compareToSeconds) {  
 return 1;  
 } else if (thisSeconds < compareToSeconds) {  
 return -1;  
 } else {  
 return 0;  
 }  
 }  
  
 *@Override*  
 public String toString() {  
 char[] array = new char[12];  
 array[0] = ***DAY\_NAMES***[day].charAt(0);  
 array[1] = ***DAY\_NAMES***[day].charAt(1);  
 array[2] = ***DAY\_NAMES***[day].charAt(2);  
 array[3] = ' ';  
 array[4] = (char) ('0' + hour / 10);  
 array[5] = (char) ('0' + hour % 10);  
 array[6] = ':';  
 array[7] = (char) ('0' + minute / 10);  
 array[8] = (char) ('0' + minute % 10);  
 array[9] = ':';  
 array[10] = (char) ('0' + second / 10);  
 array[11] = (char) ('0' + second % 10);  
 return new String(array);

}  
  
 private boolean isNegative(int number) {  
 return number < 0;  
 }  
  
}

* 1. Gränssnittet AlarmType

package se.hig.thlu.alarm;  
  
import se.hig.thlu.time.TimeType;  
  
public interface AlarmType {  
 public void setActive(boolean active);  
 public boolean isActive();  
 public void setTime(TimeType time);  
 public TimeType getTime();  
 public void addAlarmAction(AlarmActionType alarmAction);  
 public void doAlarm();  
}

* 1. Klassen Alarm

package se.hig.thlu.alarm;  
  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Objects;  
  
import se.hig.thlu.time.TimeType;  
  
public class Alarm implements AlarmType {  
 private boolean active;  
 private TimeType time;  
 private List<AlarmActionType> alarmActions = new LinkedList<AlarmActionType>();  
  
 public Alarm(TimeType time, AlarmActionType alarmAction) {  
 setTime(time);  
 addAlarmAction(alarmAction);  
 active = true;  
 }  
  
 @Override  
 public void setActive(boolean active) {  
 this.active = active;  
 if (active == false) {  
 alarmActions.forEach(alarmAction -> alarmAction.alarmDeactivated());  
 }  
 }  
  
 @Override  
 public boolean isActive() {  
 return active;  
 }  
  
 @Override  
 public void setTime(TimeType time) {  
 Objects.requireNonNull(time, "Time cannot be null.");  
 this.time = time;  
 }  
  
 @Override  
 public TimeType getTime() {  
 return time;  
 }  
  
 @Override  
 public void addAlarmAction(AlarmActionType alarmAction) {  
 Objects.requireNonNull(alarmAction, "AlarmAction cannot be null.");  
 alarmActions.add(alarmAction);  
 }  
  
 @Override  
 public void doAlarm() {  
 if (active)  
 for (AlarmActionType alarmAction : alarmActions)  
 alarmAction.alarmActivated();  
 }  
  
 public String toString() {  
 return this.time.toString();  
 }  
}

* 1. Klassen AlarmManager

package se.hig.thlu.alarm;  
  
import java.util.Collection;  
import java.util.HashMap;  
import java.util.Objects;  
  
import se.hig.thlu.time.TimeType;  
  
public class AlarmManager {  
 private HashMap<String, AlarmType> map = new HashMap<String, AlarmType>();  
  
 public void addAlarm(AlarmType alarm) {  
 Objects.requireNonNull(alarm, "Alarm cannot be null.");  
 map.put(alarm.toString(), alarm);  
 }  
  
 public void removeAlarm(AlarmType alarm) {  
 Objects.requireNonNull(alarm, "Alarm cannot be null.");  
 map.remove(alarm.toString());  
 }  
  
 public Collection<AlarmType> getAlarms() {  
 return map.values();  
 }  
  
 public void checkForAlarm(TimeType time) {  
 Objects.requireNonNull(time, "Time cannot be null.");  
 AlarmType alarm = map.get(time.toString());  
 if (alarm != null && alarm.isActive())  
 alarm.doAlarm();  
 }  
}

* 1. Gränssnittet AlarmActionType

package se.hig.thlu.alarm;  
  
public interface AlarmActionType {  
 public void alarmActivated();  
 public void alarmDeactivated();  
}

* 1. Klassen PrintAlarmActionType

package se.hig.thlu.alarm;  
  
public class PrintAlarmAction implements AlarmActionType {  
  
 @Override  
 public void alarmActivated() {  
 System.out.println("Alarm activated!");  
 }  
  
 @Override  
 public void alarmDeactivated() {  
 System.out.println("Alarm deactivated!");  
 }  
  
}

* 1. Gränssnittet ClockType

package se.hig.thlu.clock;  
  
import se.hig.thlu.time.TimeType;  
  
public interface ClockType {  
 public void tickTock();  
 public void setTime(TimeType time);  
 public TimeType getTime();  
 public void startClock(PrintTime printTime);  
 public void stopClock();  
 public void resetClock();  
 public String toString();  
}

* 1. Gränssnittet AlarmClockType

package se.hig.thlu.clock;  
  
import java.util.Collection;  
  
import se.hig.thlu.alarm.AlarmType;  
  
public interface AlarmClockType extends ClockType {  
 public void addAlarm(AlarmType alarm);  
 public void removeAlarm(AlarmType alarm);  
 public Collection<AlarmType> getAlarms();  
}

* 1. Klassen WeekClock

package se.hig.thlu.clock;  
  
import java.util.Objects;  
import java.util.Timer;  
import java.util.TimerTask;  
  
import se.hig.thlu.counter.Counter24;  
import se.hig.thlu.counter.Counter60;  
import se.hig.thlu.counter.Counter7;  
import se.hig.thlu.counter.SettableCounterType;  
import se.hig.thlu.time.Time;  
import se.hig.thlu.time.TimeType;  
  
public class WeekClock implements ClockType {  
  
 protected TimeType time;  
 protected Timer timer;  
 protected final SettableCounterType dayCounter = new Counter7();  
 protected final SettableCounterType hourCounter = new Counter24(dayCounter);  
 protected final SettableCounterType minuteCounter = new Counter60(hourCounter);  
 protected final SettableCounterType secondCounter = new Counter60(minuteCounter);  
  
 public WeekClock() {  
 setTime(new Time(0, 0, 0, 0));  
 }  
  
 public WeekClock(TimeType time) {  
 setTime(time);  
  
 dayCounter.setCount(time.getDay());  
 hourCounter.setCount(time.getHour());  
 minuteCounter.setCount(time.getMinute());  
 secondCounter.setCount(time.getSecond());  
 }  
  
 @Override  
 public void tickTock() {  
 secondCounter.count();  
 time.setDay(dayCounter.getCount());  
 time.setHour(hourCounter.getCount());  
 time.setMinute(minuteCounter.getCount());  
 time.setSecond(secondCounter.getCount());  
 }  
  
 @Override  
 public void setTime(TimeType time) {  
 this.time = Objects.requireNonNull(time);  
 }  
  
 @Override  
 public TimeType getTime() {  
 return time;  
 }  
  
 @Override  
 public void startClock(PrintTime printTime) {  
 if (timer == null) {  
 timer = new Timer();  
 timer.scheduleAtFixedRate(new TimerTask() {  
 public void run() {  
 tickTock();  
 if (printTime == PrintTime.YES) {  
 System.out.println(getTime().toString());  
 }  
 }  
 }, 0, 1000);  
 }  
 }  
  
 @Override  
 public void stopClock() {  
 if (timer != null) {  
 timer.cancel();  
 timer = null;  
 }  
 }  
  
 @Override  
 public void resetClock() {  
 stopClock();  
 dayCounter.reset();  
 hourCounter.reset();  
 minuteCounter.reset();  
 secondCounter.reset();  
 }  
  
}

* 1. Klassen WeekAlarmClock

package se.hig.thlu.clock;  
  
import java.util.Collection;  
import java.util.Objects;  
  
import se.hig.thlu.alarm.AlarmManager;  
import se.hig.thlu.alarm.AlarmType;  
import se.hig.thlu.time.Time;  
import se.hig.thlu.time.TimeType;  
  
public class WeekAlarmClock extends WeekClock implements AlarmClockType {  
  
 private final AlarmManager alarmManager = new AlarmManager();  
  
 public WeekAlarmClock(TimeType time, AlarmType alarm) {  
 super(time);  
 addAlarm(alarm);  
 }  
  
 public WeekAlarmClock(AlarmType alarm) {  
 this(new Time(0, 0, 0, 0), alarm);  
 }  
  
 @Override  
 public void tickTock() {  
 super.tickTock();  
 alarmManager.checkForAlarm(time);  
 }  
  
 @Override  
 public void addAlarm(AlarmType alarm) {  
 Objects.requireNonNull(alarm);  
 alarmManager.addAlarm(alarm);  
 }  
  
 @Override  
 public void removeAlarm(AlarmType alarm) {  
 alarmManager.removeAlarm(alarm);  
 }  
  
 @Override  
 public Collection<AlarmType> getAlarms() {  
 return alarmManager.getAlarms();  
 }  
}

* 1. Enum PrintTime

package se.hig.thlu.clock;  
  
public enum PrintTime {  
 YES, NO  
}