CS120: Designing Classes

W. Michael Petullo

University of Wisconsin–La Crosse

As of November 1, 2021

Designing Classes: Introduction

- A class defines a new type of object.
- A class definition is a template for objects: attributes the objects have along with methods that can operate on them.
- Every object is an instance of some class.
- The `new` operator instantiates an object, creating a new instance of a class.

Define a new class to encapsulate related data in an object that can be treated as a single unit.

Designing Classes: `Time`

```
public class Time {
        private int hour;
        private int minute;
        private double second;
}
```

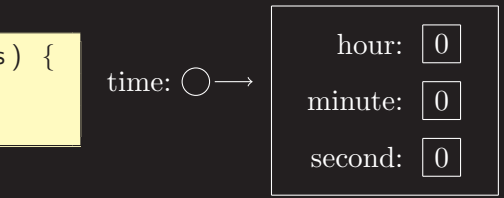
Class `time` has three integer attributes: `hour`, `minute`, and `second`. Each attribute is `private`, meaning users of the class cannot directly access them.

Designing Classes: `Time` Constructor

```
public Time() {
    this.hour = 0;
    this.minute = 0;
    this.second = 0;
}
```

- The name of the constructor matches the name of the class.
- Constructors have not return type.
- Omit the keyword `static`.
- `this` refers to object we are creating.

```
public static void main(String[] args) {
    Time time = new Time();
}
```

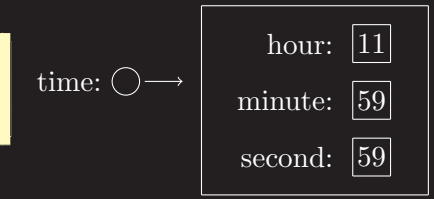


Designing Classes: `Time` Value Constructor

```
public Time(int hour, int minute, double second) {
    this.hour = hour;
    this.minute = minute;
    this.second = second;
}
```

- Like all methods, we can overload constructors.
- Notice how use of `this` is necessary to describe to which `hour`, `minute`, and `second` we refer.

```
public static void main(String[] args) {
    Time time = new Time(11, 59, 59);
}
```



Designing Classes: Impact of `private`

```
public class TimeClient {
    public static void main(String[] args) {
        Time time = new Time(11, 59, 59);
        System.out.println(time.hour);
    }
}
```

- Compiler error! The `hour` attribute is `private`, so `TimeClient` cannot access it.
- Options:
  - Make the attributes `public`.
  - Provide methods to access attributes.
  - Forbid other classes from accessing.

Designing Classes: Getters

```
public int getHour() {
    return this.hour;
}

public int getMinute() {
    return this.minute;
}

public double getSecond() {
    return this.second;
}
```

Providing only getters leaves `Time` as an immutable object.

```
public static void main(String[] args) {
    Time time = new Time(11, 59, 59);
    System.out.println(time.getHour());
}
```

Designing Classes: Setters

```
public int setHour(int hour) {
    this.hour = hour;
}

public int setMinute(int minute) {
    this.minute = minute;
}

public double setSecond(double second) {
    this.second = second;
}
```

Providing setters turns `Time` into a mutable object.

```
public static void main(String[] args) {
    Time time = new Time(11, 59, 59);
    time.setHour(10);
    System.out.println(time.getHour());
}
```

Designing Classes: toString

```
public String toString() {
    return String.format("%02d:%02d:%04.1f\n", this.hour,
        this.minute, this.second);
}
```

- Omit the `static` keyword when writing an instance method, which acts on a particular object (`this`).
- The customary `toString` method renders an object as a string.

```
public static void main(String[] args) {
    Time time = new Time(11, 59, 59);
    time.setHour(10);
    System.out.println(time.toString());
}
```

Designing Classes: `equals`

```
public boolean equals(Time that) {
    final double DELTA = 0.001;
    return this.hour == that.hour
        && this.minute == that.minute
        && Math.abs(this.second - that.second) < DELTA;
}
```

- An `equals` method overcomes the limitation of comparing references to objects.
- The `double` type is an approximation, so take this into account when comparing.

```
public static void main(String[] args) {
    Time time1 = new Time(11, 59, 59);
    Time time2 = new Time(11, 59, 59);
    System.out.println(time1.equals(time2));
}
```

Designing Classes: toString

```
public Time add(Time that) {
    this.second += that.second;
    this.minute += that.minute;
    this.hour += that.hour;

    if (this.second >= 60) {
        this.second -= 60;
        this.minute += 1;
    }

    if (this.minute >= 60) {
        this.minute -= 60;
        this.hour += 1;
    }

    if (this.hour >= 24) {
        this.hour -= 24;
    }
}
```

```
public static void
    main(String[] args) {
    Time time1 =
        new Time(11, 45, 45);
    Time time2 =
        new Time(11, 45, 45);
    time1.add(time2);
    System.out.println(time1);
}
```

Designing Classes: Assignments

Graded Homework Aquinas: triangle2 in Java

Ungraded Labs Aquinas: guest and uml in Java

Reading Read Chapter 11

`https://www.flyn.org/courses/cs120-2021-fall/schedule`