SAT: Appropriate Constructs

Improve the code quality of the following programs:

1.

```
class CruiseControl {

    private double targetSpeedKmh;

    void setPreset(int speedPreset) {
        if (speedPreset == 2) {
            setTargetSpeedKmh(16944);
        } else if (speedPreset == 1) {
            setTargetSpeedKmh(7667);
        } else if (speedPreset == 0) {
            setTargetSpeedKmh(0);
        }
    }

    void setTargetSpeedKmh(double speed) {
        targetSpeedKmh = speed;
    }
}
```

```
class CruiseControl {

    private double targetSpeedKmh;

    void setPreset(SpeedPreset speedPreset) {
        Objects.requireNonNull(speedPreset);

        setTargetSpeedKmh(speedPreset.speedKmh);
    }

    void setTargetSpeedKmh(double speedKmh) {
        targetSpeedKmh = speedKmh;
    }
}
enum SpeedPreset {
    STOP(0), PLANETARY_SPEED(7667), CRUISE_SPEED(16944);

    final double speedKmh;

    SpeedPreset(double speedKmh) {
        this.speedKmh = speedKmh;
    }
}
```

2.

```
class LaunchChecklist {

    List<String> checks = Arrays.asList("Cabin Pressure",
                                        "Communication",
                                        "Engine");

    Status prepareForTakeoff(Commander commander) {
        for (int i = 0; i < checks.size(); i++) {
            boolean shouldAbortTakeoff = commander.isFailing(checks.get(i));
            if (shouldAbortTakeoff) {
                return Status.ABORT_TAKE_OFF;
            }
        }
        return Status.READY_FOR_TAKE_OFF;
    }
}
```

```
class LaunchChecklist {

    List<String> checks = Arrays.asList("Cabin Pressure",
                                        "Communication",
                                        "Engine");

    Status prepareForTakeoff(Commander commander) {
        for (String check : checks) {
            boolean shouldAbortTakeoff = commander.isFailing(check);
            if (shouldAbortTakeoff) {
                return Status.ABORT_TAKE_OFF;
            }
        }
        return Status.READY_FOR_TAKE_OFF;
    }
}
```

3.

```
class Inventory {

    private List<Supply> supplies = new ArrayList<>();

    void disposeContaminatedSupplies() {
        for (Supply supply : supplies) {
            if (supply.isContaminated()) {
                supplies.remove(supply);
            }
        }
    }
}
```

```
class Inventory {

    private List<Supply> supplies = new ArrayList<>();

    void disposeContaminatedSupplies() {
        Iterator<Supply> iterator = supplies.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().isContaminated()) {
                iterator.remove();
            }
        }
    }
}
```

4.

```
class Inventory {

    private List<Supply> supplies = new ArrayList<>();

    int getQuantity(Supply supply) {
        if (supply == null) {
            throw new NullPointerException("supply must not be null");
        }

        int quantity = 0;
        for (Supply supplyInStock : supplies) {
            if (supply.equals(supplyInStock)) {
                quantity++;
            }
        }

        return quantity;

    }
}
```

```
class Inventory {

    private List<Supply> supplies = new ArrayList<>();

    int getQuantity(Supply supply) {
        Objects.requireNonNull(supply, "supply must not be null");

        return Collections.frequency(supplies, supply);
    }
}
```

5.

```
class Mission {

    Logbook logbook;
    LocalDate start;

    void update(String author, String message) {
        LocalDate today = LocalDate.now();
        String month = String.valueOf(today.getMonthValue());
        String formattedMonth = month.length() < 2 ? "0" + month : month;
        String entry = author.toUpperCase() + ": [" + formattedMonth + "-" +
                today.getDayOfMonth() + "-" + today.getYear() + "](Day " +
                (ChronoUnit.DAYS.between(start, today) + 1) + ")> " +
                message + System.lineSeparator();
        logbook.write(entry);
    }
}
```

```
class Mission {

    Logbook logbook;
    LocalDate start;

    void update(String author, String message) {
        final LocalDate today = LocalDate.now();
        String entry = String.format("%S: [%tm-%<te-%<tY](Day %d)> %s%n",
                author, today,
                ChronoUnit.DAYS.between(start, today) + 1, message);
        logbook.write(entry);
    }
}
```