A Tale of Two Cities

Magnus Madsen
Software Architecture Project
Winner Strategy

You are asked to implement:

```java
public Player getWinner(Collection<City> cities);
```

where a player has won if he has conquered all cities, i.e. RED has won if BLU controls no cities (and vice versa)

Real world issue: What if there are no cities?
=> Implementation dependent
Think, but don't speak

How would you implement this in Java?
In your favorite language?

**RED** has won if **BLU**
controls no cities
(and vice versa)
public Player getWinner(GameImpl game) {
    Player winner = null;
    Player lastOwner = null;
    for (CityImpl c : game.getCities()) {
        if (lastOwner == null) {
            lastOwner = c.getOwner();
            winner = lastOwner;
        } else {
            if (!lastOwner.equals(c.getOwner())) {
                winner = null;
                break;
            }
        }
    }
    return winner;
}
public Player getWinner(GameImpl game) {
    List<Player> players = new ArrayList<Player>();
    for (CityImpl c : game.getCitiesIterable()) {
        if (!players.contains(c.getOwner())) {
            players.add(c.getOwner());
            if (players.size() >= 2) break;
        }
    }
    if (players.size() == 1) return players.get(0);
    else return null;
}
```java
public Player computeWinner(GameImpl game) {
    Player assumedWinner = null;
    for (CityImpl c : game.getCities().values()) {
        if (assumedWinner == null) {
            assumedWinner = c.getOwner();
        } else {
            if (!c.getOwner().equals(assumedWinner))
                return null;
        }
    }
    return assumedWinner;
}
```
Solution IV

```java
public Player computeWinner(GameImpl game) {
    int red = 0;
    int blue = 0;
    for (CityImpl c : game.getCities().values()) {
        if (c.getOwner() == Player.RED) {
        } else {
            blue++;
        }
    }
    if (blue == 0) {
        return Player.RED;
    }
    if (red == 0) {
        return Player.BLUE;
    }
    return null;
}
```
def getWinner(cities: ParSet[City]): City = {
    val redCity = cities.exists(c => c.isRed);
    val blueCity = cities.exists(c => c.isBlue);
    if (!blueCity)
        Player.Red
    else if (!redCity)
        Player.Blue
    else
        null
}
def getWinner(cities: ParSet[City]):
    if (cities.isEmpty)
        return null;
    return cities.reduce((c1, c2) =>
        if (c1 == c2) c1 else null
    );

def getWinner(cities: ParSet[City]):
    if (cities.isEmpty)
        return null;
    val c1 = cities.head.color;
    if (cities.exists(c2 => c1 != c2.color))
        null
    else
        c1

Summary

Interesting properties:

– obvious implementation (i.e. correctness)?
– n > 2 players?
– early termination?
– single pass?
– embarrassingly parallel?

A variety of interesting solutions:

– mutating, accumulating, counting, reducing, ...