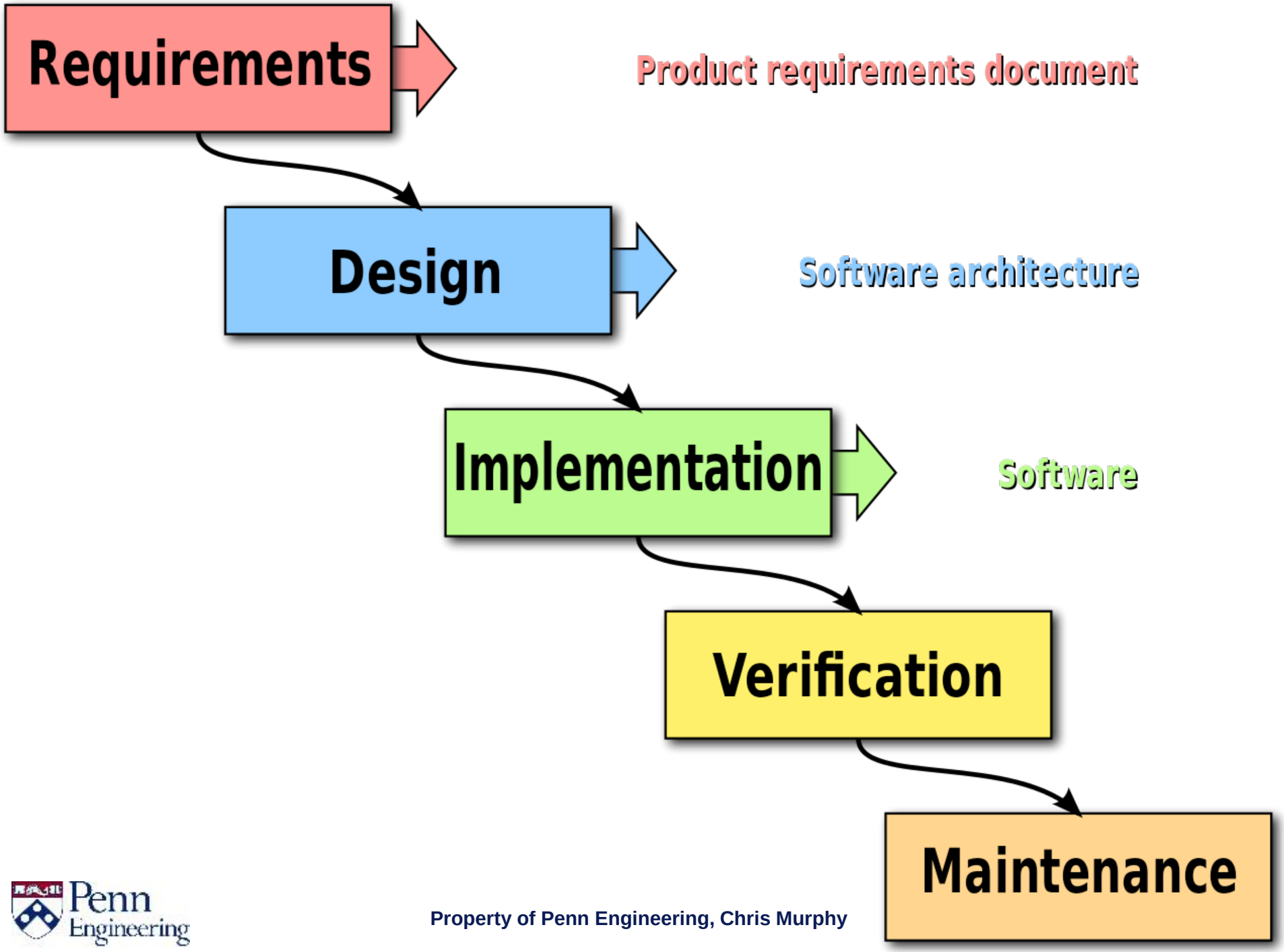
SD2x4.1

Motivating Software Maintenance

Chris

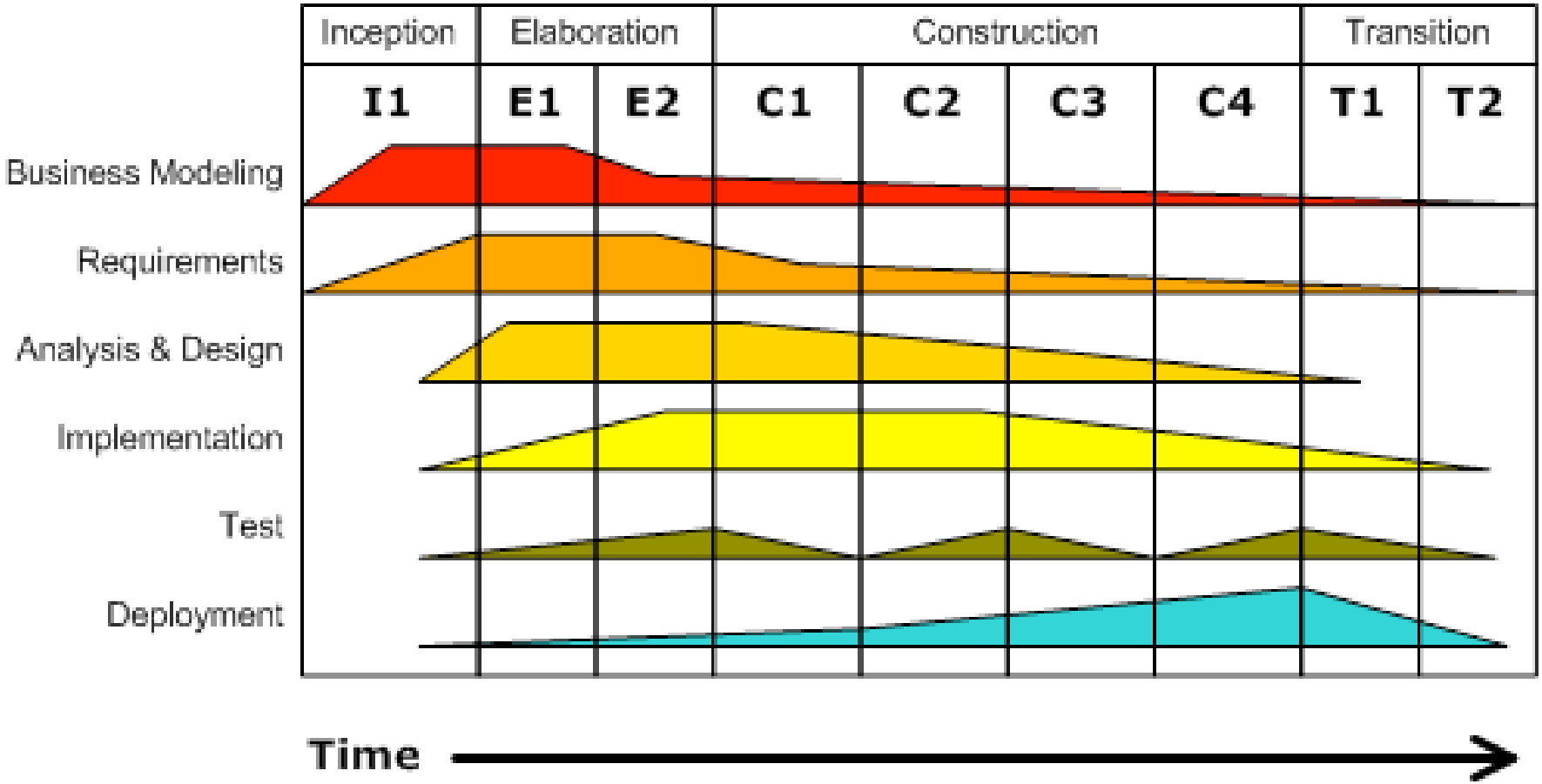
What’s a software process model?

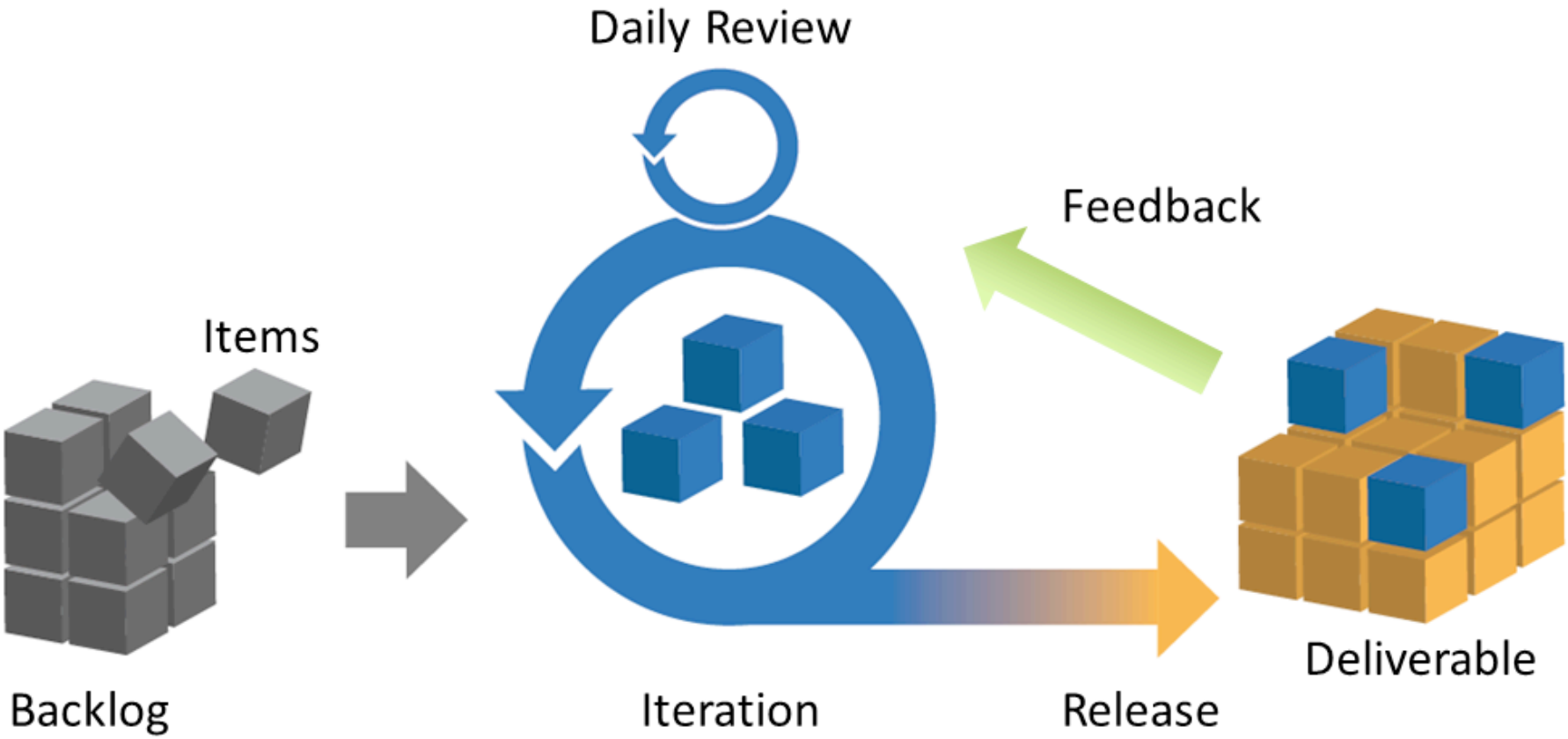

Requirements
Product requirements document
Design
Software architecture
Implementation
Software
Verification
Maintenance

RUP®
Rational Unified Process®

Iterative Development

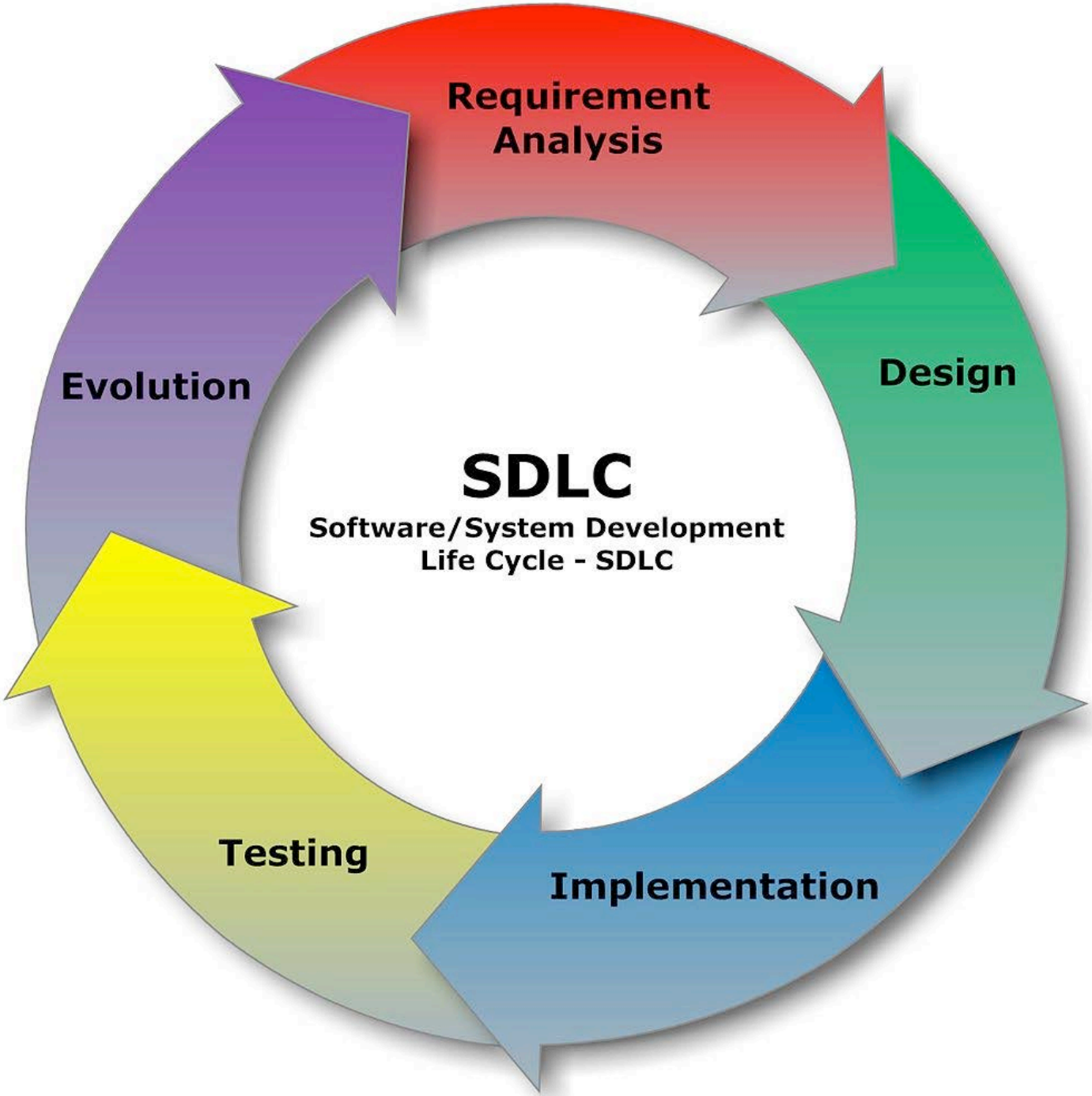
Business value is delivered incrementally in time-boxed cross-discipline iterations.





Plan ***Collaborate*** ***Deliver***

Agile Project Management: Iteration


Requirement Analysis
Design
Implementation
Testing
Evolution
SDLC
Software/System Development Life Cycle - SDLC

What is software maintenance?

Software Maintenance

- The process of **modifying** software after it has been deployed/delivered
- Anywhere from 40-90% of the time/cost of developing software is spent on maintenance
- Identify “defects,” fix them, make sure you haven’t broken anything else
- Nowadays, a constant, ongoing process of improving the software’s **quality**

Software Quality

- **Functionality:** satisfies stated or implied needs
- **Reliability:** maintains a level of performance in a given environment for a given period of time
- **Usability:** ease with which user can understand, learn, and operate the software
- **Efficiency:** limits the amount/number of resources used
- **Portability:** ease with which software can be transferred from one environment to another
- **Security:** level of confidentiality and integrity provided by the software
- **Maintainability:** ease with which programmer can analyze, modify, and improve the code

Software Quality

- **Functionality:** satisfies stated or implied needs
- **Reliability:** maintains a level of performance in a given environment for a given period of time
- **Usability:** ease with which user can understand, learn, and operate the software
- **Efficiency:** limits the amount/number of resources used
- **Portability:** ease with which software can be transferred from one environment to another
- **Security:** level of confidentiality and integrity provided by the software
- **Maintainability:** ease with which programmer can analyze, modify, and improve the code

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

SD2x4.2

Analyzability

Chris

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

Why is it important that someone can read/understand your code?

Why do people need to read/understand code?

- Fix bugs
- Add features
- Understand how you did something
- Improve other aspects of software quality (efficiency, security, etc.)

What are "Readability" and "Understandability"?

What affects Readability?

Why is this code hard to read?

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

Why is the second one easier to read?

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

1. Use of whitespace

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

1. Use of whitespace

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

2. Identifier length

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

2. Identifier length

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

3. Use of dictionary words

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

3. Use of dictionary words

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

4. Variations between identifiers

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

4. Variations between identifiers

```
int da_D4eds31f=0;
for(int da__D4es31=0;da__D4es31<dda_D4efs31.length;da__D4es31+=1)
if(dda_D4efs31[da__D4es31]<0)da_D4eds31f=da_D4eds31f+1;
```

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

Readability

- The ease with which the reader can identify and differentiate tokens and their **syntactic purpose** (role in the statement)

- What affects it?
  - Use of whitespace
  - Identifier length
  - Use of dictionary words
  - Variation between identifiers

What affects Understandability?

How can we make this easier to understand?

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

1. Use meaningful identifier names

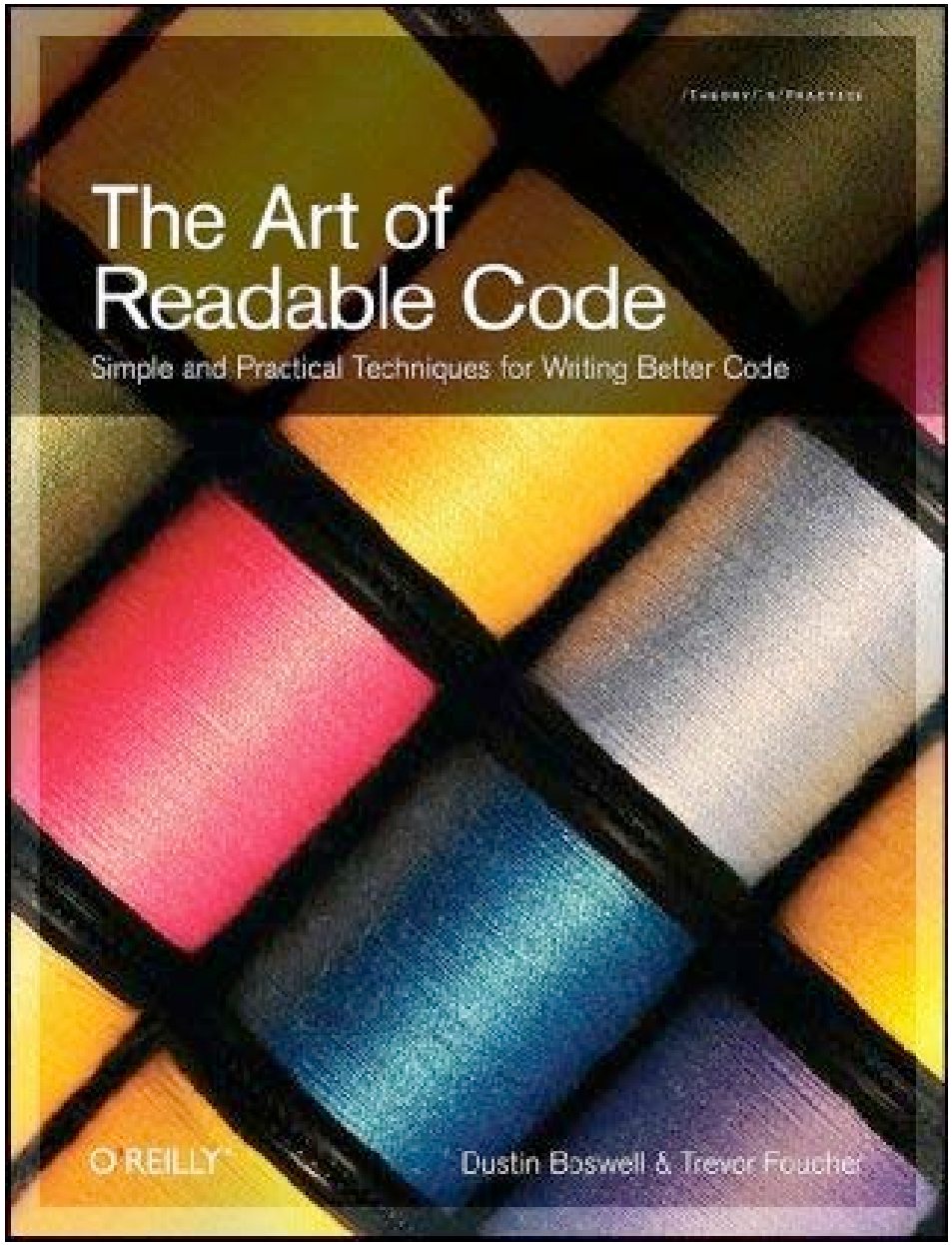
```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

```
int count = 0;
for (int i = 0; i < values.length; values += 1)
if (values[i] < 0) count = count + 1;
```

1. Use meaningful identifier names

```
int ant = 0;
for (int bat = 0; bat < cow.length; bat += 1)
if (cow[bat] < 0) ant = ant + 1;
```

```
int count = 0;
for (int i = 0; i < values.length; values += 1)
if (values[i] < 0) count = count + 1;
```


The Art of
Readable Code
Simple and Practical Techniques for Writing Better Code
O'REILLY
Dustin Boswell & Trevor Foucher

What's a "meaningful" name?

- Unambiguous:
  - "select" instead of "filter"
  - "first"/"last" instead of "start"/"stop"
  - "shouldReadPassword" instead of "readPassword"

- use specific words: "download" or "calculate" instead of "get"
- avoid generic names like "temp", "x", etc.
- attach details, e.g. "timeInMs" or "rawData"

2. Use indentation

```
int count = 0;
for (int i = 0; i < values.length; values += 1)
if (values[i] < 0) count = count + 1;
```

```
int count = 0;
for (int i = 0; i < values.length; i++)
     if (values[i] < 0)
         count++;
```

2. Use indentation

```
int count = 0;
for (int i = 0; i < values.length; values += 1)
if (values[i] < 0) count = count + 1;
```

```
int count = 0;
for (int i = 0; i < values.length; i++)
     if (values[i] < 0)
         count++;
```

3. Use punctuation

```
int count = 0;
for (int i = 0; i < values.length; i++)
     if (values[i] < 0)
         count++;
```

```
int count = 0;
for (int i = 0; i < values.length; i++) {
     if (values[i] < 0) {
         count++;
     }
}
```

3. Use punctuation

```
int count = 0;
for (int i = 0; i < values.length; i++)
     if (values[i] < 0)
         count++;
```

```
int count = 0;
for (int i = 0; i < values.length; i++) {
     if (values[i] < 0) {
         count++;
     }
}
```

4. Use comments as needed

```
int count = 0;
for (int i = 0; i < values.length; i++) {
     if (values[i] < 0) {
          count++;
     }
}
```

```
// Sets count to the number of negative elements in values array
int count = 0;
for (int i = 0; i < values.length; i++) {
     if (values[i] < 0) {
          count++;
     }
}
```

Why is this hard to understand?

```
while (a > b) {
    if (c % a == b + (d--)) {
        b = e + c;
        a *= -1;
    }
    else {
        while ((c == e) || (a > f)) {
            b = f % a;
            a++;
            c = a * b ;
        }
    }
}
```

1. Data complexity

```
while (a > b) {
    if (c % a == b + (d--)) {
        b = e + c;
        a *= -1;
    }
    else {
        while ((c == e) || (a > f)) {
            b = f % a;
            a++;
            c = a * b ;
        }
    }
}
```

2. Structural complexity

```
while (a > b) {
    if (c % a == b + (d--)) {
        b = e + c;
        a *= -1;
    }
    else {
        while ((c == e) || (a > f)) {
            b = f % a;
            a++;
            c = a * b ;
        }
    }
}
```

Understandability

- The ease with which the reader can identify the **semantic meaning** of code (what it does, what its purpose is)
- What affects it?
  - Readability!
  - Meaningful identifier names
  - Indentation
  - Punctuation
  - Comments
  - Complexity: data and structural
  - Adherence to coding conventions

Java code conventions

- **Class** and **interface** names:
  - nouns
  - start with capital letter
  - capitalize first letter of each internal word (“CamelCaps”)
- **Method** names:
  - verbs
  - start with lowercase letter; camelCaps
- **Field** and **local variable** names
  - nouns
  - start with lowercase letter; camelCaps

Recap: Analyzability

- **Readability:** ease with which someone can identify and differentiate tokens and their syntactic purpose

- **Understandability**: ease with which someone can identify the semantic meaning of code

- Both are affected by appearance, complexity, and adherence to convention

SD2x4.3

Refactoring

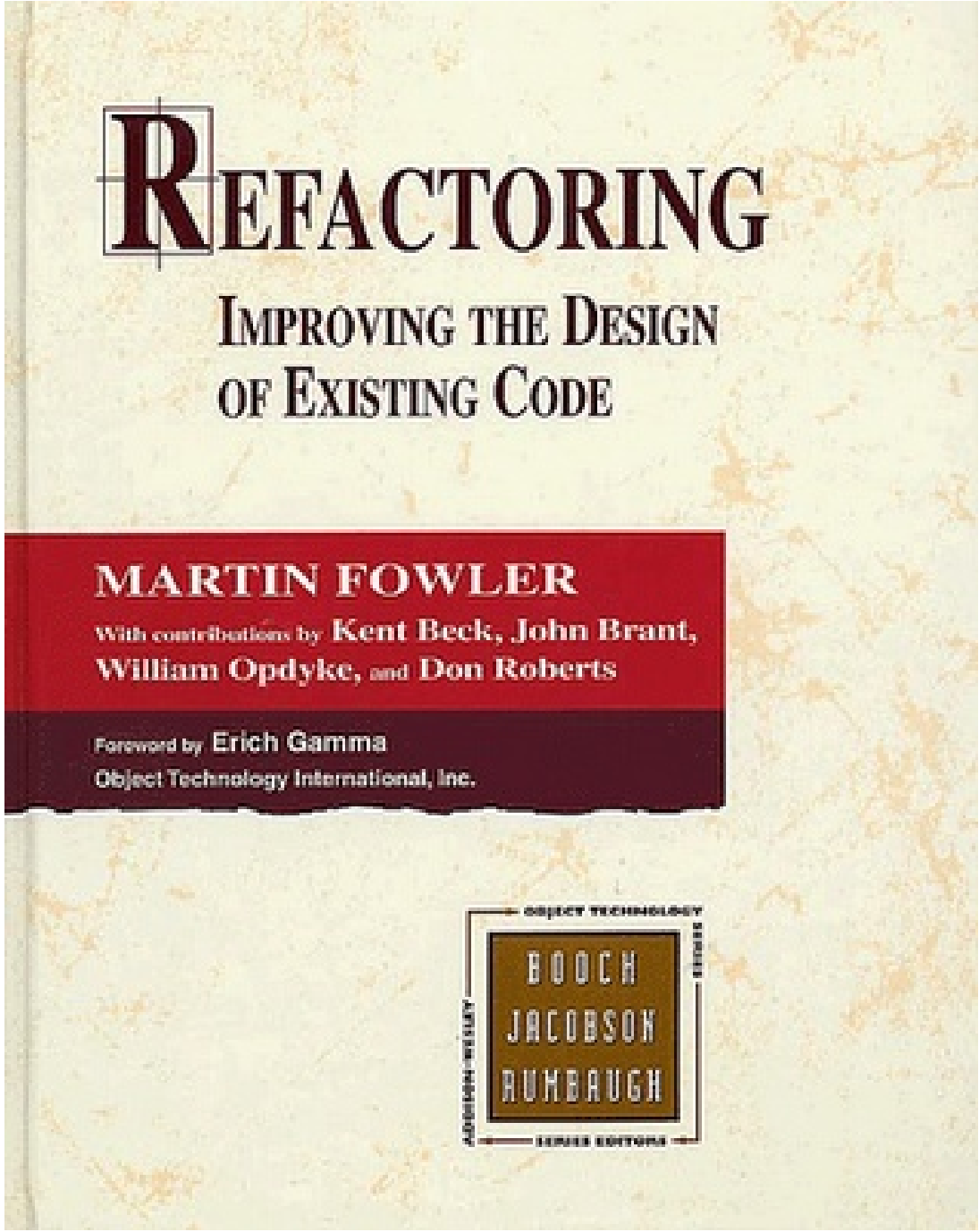
Chris

“Any fool can write code that a computer will understand.

“Any fool can write code that a computer will understand.

Good programmers write code that **humans** can understand.”

-Martin Fowler


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
OBJECT TECHNOLOGY
BOOCH
JACOBSON
RUMBAUGH
SERIES EDITORS

Goals of Refactoring

- Improve the internal quality (design) of the code
- Simplify future code changes
- Reduce the amount of code to maintain

- Improve efficiency **only** if it needs it or is obviously inefficient
- Refactoring is **not** bug fixing!

When should you refactor?

- **CONSTANTLY! "Refactor mercilessly"**

- When you see code that doesn't adhere to the intended design

- When you are making changes and realizing you're not adhering to the design

- When you want to get an understanding of the code

How to refactor?

- **CAREFULLY!**
- Run your tests after *every* change
- Make small changes before big ones
- Be careful about:
  - Introducing bugs
  - Breaking “published” interfaces
  - Unnecessarily decreasing external quality

What to refactor: “Code Smells”

- Indications of bad design
- Code that is hard to maintain (read, understand, change, etc.)
- Code that does not adhere to conventions
- Code that is unnecessary or in the “wrong place”

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> a = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      a.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : a) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    private String name;
    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      this.name = name;
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {


    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {

      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void add_player(String name) {
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void go() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++; // increment clock
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
      if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
      else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
        else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
        else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true)p.move();
        else if (p.isAlive() && p.points() > 0) p.pause();
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if(p.isAlive()&&p.points()> 0&&p.ready()==true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready() == true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready() == true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready() == true) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready()) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready()) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0 && p.ready()) {
           p.move();
        }
        else if (p.isAlive() && p.points() > 0) {
           p.pause();
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0) {
           if (p.ready()) {
               p.move();
           }
           else {
               p.pause();
           }
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0) {
           if (p.ready()) {
               p.move();
           }
           else {
               p.pause();
           }
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0) {
           if (p.ready()) {
               p.move();
           }
           else {
               p.pause();
           }
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0) {
           if (p.ready()) {
               p.move();
           }
           else {
               p.pause();
           }
        }
      }
    }
}
```

```
public class MyFunGame {

    public int clock = 0;
    protected List<Player> players = new LinkedList<Player>();

    public void addPlayer(String name) {
      players.add(new Player(name));
    }

    public void playOneStep() {
      clock++;
      for (Player p : players) {
        if (p.isAlive() && p.points() > 0) {
           if (p.ready()) {
               p.move();
           }
           else {
               p.pause();
           }
        }
      }
    }
}
```

Recap: Refactoring

- Improve the internal quality (design) of the code
- Do this carefully! Always test after each change.
- Be on the lookout for Code Smells

SD2x4.4
Duplicate code
Chris

Refactoring

- Improve the internal quality (design) of the code
- Simplify future code changes
- Reduce the amount of code to maintain

- Address problems known as **Code Smells**

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```java
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```java
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = 0;
    for (int i = 0; i < array1.length; i++) {
        sum1 += array1[i];
    }
    int sum2 = 0;
    for (int i = 0; i < array2.length; i++) {
        sum2 += array2[i];
    }
    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = 0;
    for (int i = 0; i < array1.length; i++) {
        sum1 += array1[i];
    }
    int sum2 = 0;
    for (int i = 0; i < array2.length; i++) {
        sum2 += array2[i];
    }
    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = 0;
    for (int i = 0; i < array1.length; i++) {
        sum1 += array1[i];
    }
    int sum2 = 0;
    for (int i = 0; i < array2.length; i++) {
        sum2 += array2[i];
    }
    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}
```

Duplicate Code!

Duplicate Code

- #1 most common code smell
- **understandability**: more code to read
- **changeability**: more places to change
- **testability**: more opportunities for bugs

Solution:
Extract Method!

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```java
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}

private static int calculateArraySum(int[] arr) {
   int arraySum = 0;
   for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
   }
   return arraySum;
}
```

```java
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = 0;
   for (int i = 0; i < array1.length; i++) {
      sum1 += array1[i];
   }
   int sum2 = 0;
   for (int i = 0; i < array2.length; i++) {
      sum2 += array2[i];
   }
   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}

private static int calculateArraySum(int[] arr) {
   int arraySum = 0;
   for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
   }
   return arraySum;
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = 0;
    for (int i = 0; i < array1.length; i++) {
        sum1 += array1[i];
    }
    int sum2 = 0;
    for (int i = 0; i < array2.length; i++) {
        sum2 += array2[i];
    }
    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}

private static int calculateArraySum(int[] arr) {
    int arraySum = 0;
    for (int i = 0; i < arr.length; i++) {
        arraySum += arr[i];
    }
    return arraySum;
}
```

```java
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = 0;
    for (int i = 0; i < array1.length; i++) {
        sum1 += array1[i];
    }
    int sum2 = 0;
    for (int i = 0; i < array2.length; i++) {
        sum2 += array2[i];
    }
    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}

private static int calculateArraySum(int[] arr) {
    int arraySum = 0;
    for (int i = 0; i < arr.length; i++) {
        arraySum += arr[i];
    }
    return arraySum;
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
    int sum1 = calculateArraySum(array1);



    int sum2 = calculateArraySum(array2);



    if (sum1 > sum2) {
        return array1;
    } else {
        return array2;
    }
}

private static int calculateArraySum(int[] arr) {
    int arraySum = 0;
    for (int i = 0; i < arr.length; i++) {
        arraySum += arr[i];
    }
    return arraySum;
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = calculateArraySum(array1);



   int sum2 = calculateArraySum(array2);



   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}

private static int calculateArraySum(int[] arr) {
   int arraySum = 0;
   for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
   }
   return arraySum;
}
```

```
public static int[] arrayMax(int[] array1, int[] array2) {
   int sum1 = calculateArraySum(array1);
   int sum2 = calculateArraySum(array2);

   if (sum1 > sum2) {
      return array1;
   } else {
      return array2;
   }
}




private static int calculateArraySum(int[] arr) {
   int arraySum = 0;
   for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
   }
   return arraySum;
}
```

What if the duplicate code is in two different classes?

```
public class EventLogistics {

   protected int maxCapacity;
   private int currentBudget;

   public EventLogistics() {
      maxCapacity = 0;
      currentBudget = 0;
   }

   public void updateWithProfits(int[] siteProfits) {
      for (int i = 0; i < siteProfits.length; i++) {
         currentBudget += siteProfits[i];
      }
      maxCapacity = Math.floor((float) currentBudget * 0.75);
   }
}
```

```java
public class EventLogistics {

   protected int maxCapacity;
   private int currentBudget;

   public EventLogistics() {
      maxCapacity = 0;
      currentBudget = 0;
   }

   public void updateWithProfits(int[] siteProfits) {
      for (int i = 0; i < siteProfits.length; i++) {
         currentBudget += siteProfits[i];
      }
      maxCapacity = Math.floor((float) currentBudget * 0.75);
   }
}
```

```java
public class EventLogistics {

   protected int maxCapacity;
   private int currentBudget;

   public EventLogistics() {
      maxCapacity = 0;
      currentBudget = 0;
   }

   public void updateWithProfits(int[] siteProfits) {
      for (int i = 0; i < siteProfits.length; i++) {
         currentBudget += siteProfits[i];
      }
      maxCapacity = Math.floor((float) currentBudget * 0.75);
   }
}
```

```java
public class GrantLogistics {

   private int currentBudget;

   public GrantLogistics(int amountAvailable) {
      currentBudget = amountAvailable;
   }

   public boolean fulfillAllRequests(int[] individualRequests) {
      int totalAmountRequested = 0;
      for (int i = 0; i < individualRequests.length; i++) {
         totalAmountRequested += individualRequests[i];
      }
      return totalAmountRequested <= currentBudget;
   }
}
```

```java
public class GrantLogistics {

   private int currentBudget;

   public GrantLogistics(int amountAvailable) {
      currentBudget = amountAvailable;
   }

   public boolean fulfillAllRequests(int[] individualRequests) {
      int totalAmountRequested = 0;
      for (int i = 0; i < individualRequests.length; i++) {
         totalAmountRequested += individualRequests[i];
      }
      return totalAmountRequested <= currentBudget;
   }
}
```

```java
public class GrantLogistics {

   private int currentBudget;

   public GrantLogistics(int amountAvailable) {
      currentBudget = amountAvailable;
   }

   public boolean fulfillAllRequests(int[] individualRequests) {
      int totalAmountRequested = 0;
      for (int i = 0; i < individualRequests.length; i++) {
         totalAmountRequested += individualRequests[i];
      }
      return totalAmountRequested <= currentBudget;
   }
}
```

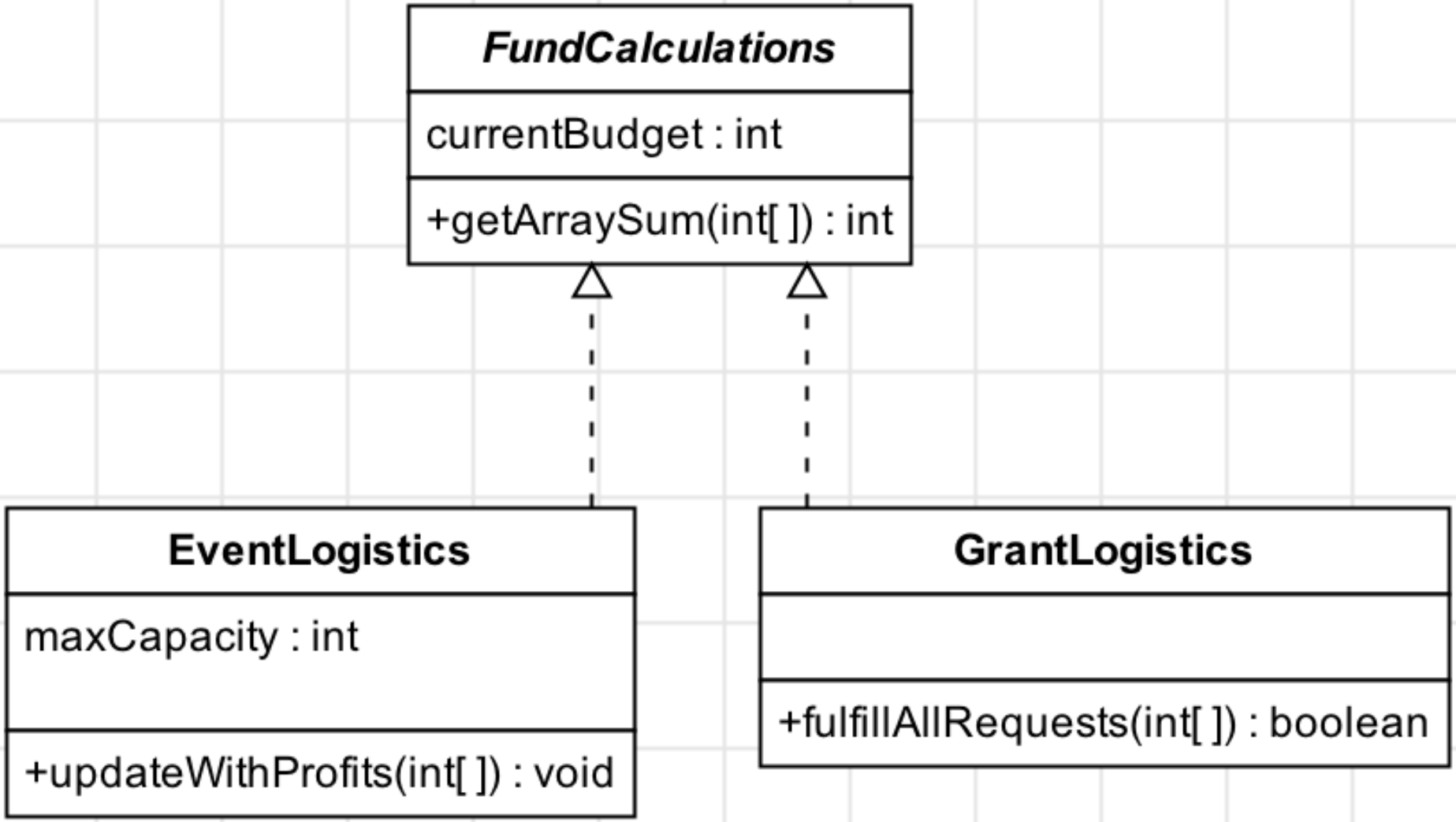
Solution:
Extract Superclass!

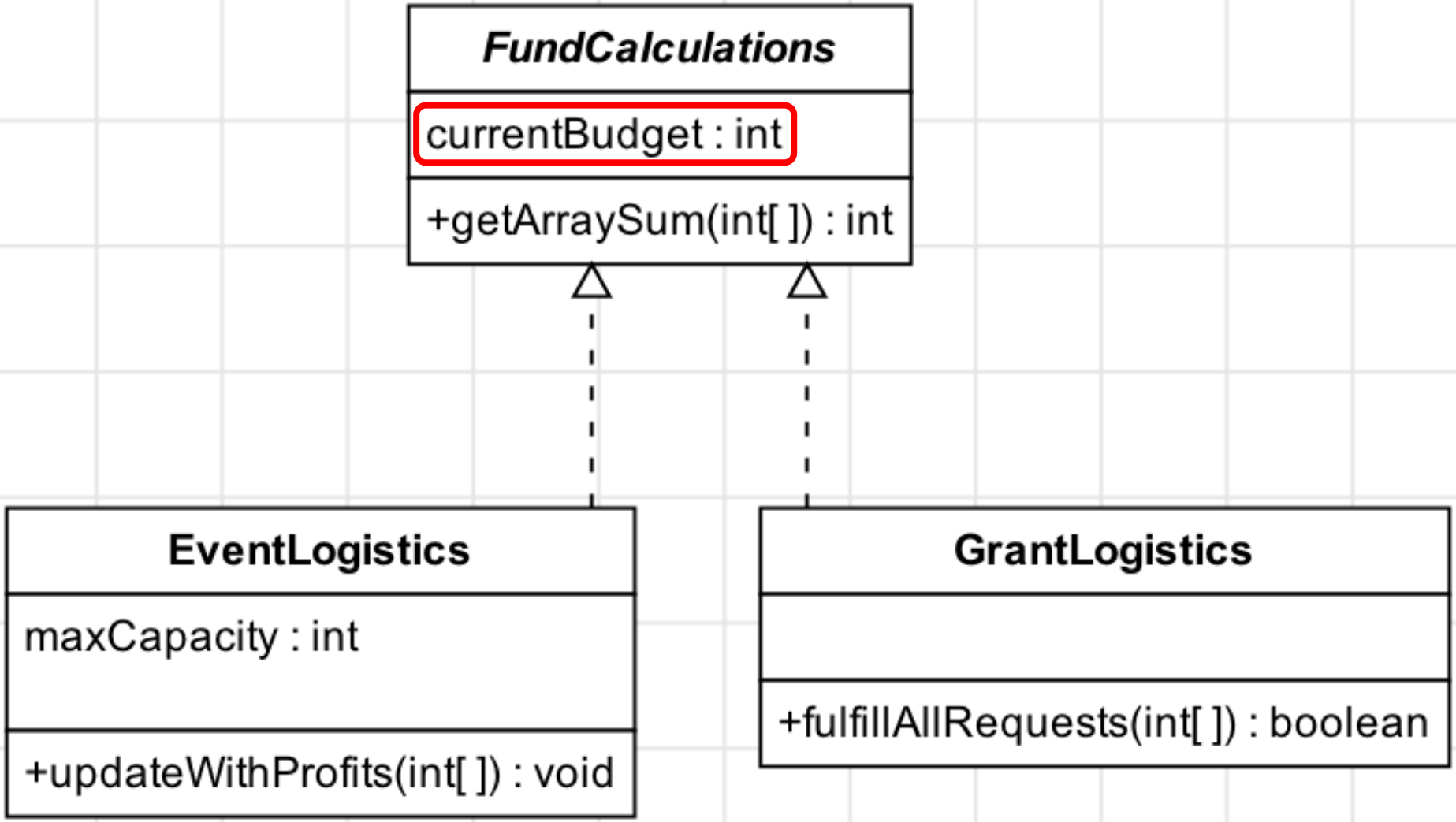
EventLogistics
maxCapacity : int currentBudget : int
+updateWithProfits(int[ ]) : void

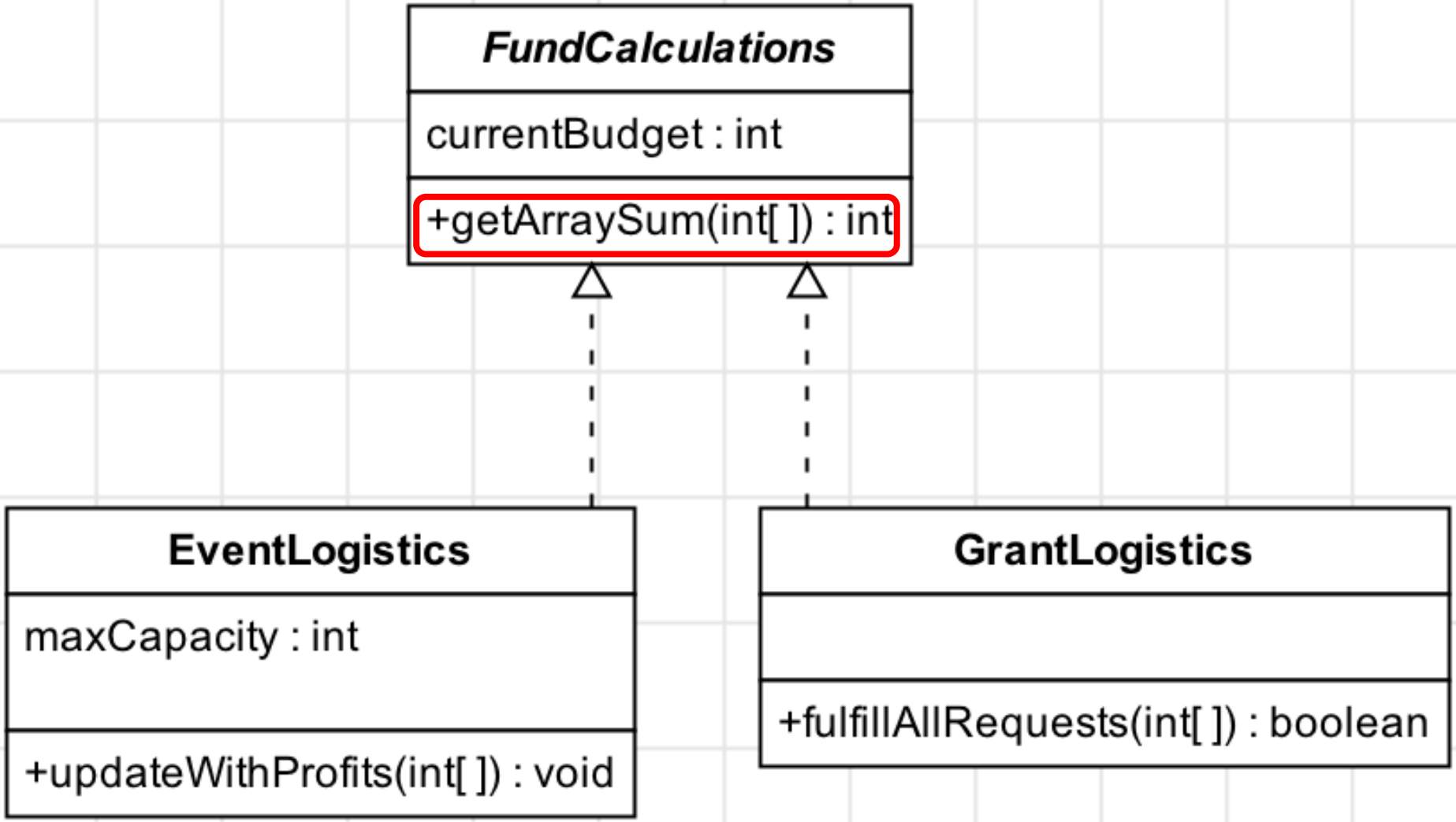
GrantLogistics
currentBudget : int
+fulfillAllRequests(int[ ]) : boolean

EventLogistics
maxCapacity : int currentBudget : int
+updateWithProfits(int[ ]) : void

GrantLogistics
currentBudget : int
+fulfillAllRequests(int[ ]) : boolean


FundCalculations
currentBudget : int
+getArraySum(int[ ]) : int
EventLogistics
maxCapacity : int
+updateWithProfits(int[ ]) : void
GrantLogistics
+fulfillAllRequests(int[ ]) : boolean


FundCalculations
currentBudget : int
+getArraySum(int[ ]) : int
EventLogistics
maxCapacity : int
+updateWithProfits(int[ ]) : void
GrantLogistics
+fulfillAllRequests(int[ ]) : boolean


FundCalculations
currentBudget : int
+getArraySum(int[ ]) : int
EventLogistics
maxCapacity : int
+updateWithProfits(int[ ]) : void
GrantLogistics
+fulfillAllRequests(int[ ]) : boolean

Extract Superclass

```
public abstract class FundCalculations {

   protected int currentBudget;

   public FundCalculations() {
      currentBudget = 0;
   }

   protected int getArraySum(int[] arr) {
     int arraySum = 0;
     for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
     }
     return arraySum;
   }
}
```

Extract Superclass

```
public abstract class FundCalculations {

   protected int currentBudget;

   public FundCalculations() {
      currentBudget = 0;
   }

   protected int getArraySum(int[] arr) {
     int arraySum = 0;
     for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
     }
     return arraySum;
   }
}
```

Extract Superclass

```
public abstract class FundCalculations {

   protected int currentBudget;

   public FundCalculations() {
      currentBudget = 0;
   }

   protected int getArraySum(int[] arr) {
     int arraySum = 0;
     for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
     }
     return arraySum;
   }
}
```

Extract Superclass

```
public abstract class FundCalculations {

   protected int currentBudget;

   public FundCalculations() {
      currentBudget = 0;
   }

   protected int getArraySum(int[] arr) {
     int arraySum = 0;
     for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
     }
     return arraySum;
   }
}
```

Extract Superclass

```
public abstract class FundCalculations {

   protected int currentBudget;

   public FundCalculations() {
      currentBudget = 0;
   }

   protected int getArraySum(int[] arr) {
     int arraySum = 0;
     for (int i = 0; i < arr.length; i++) {
      arraySum += arr[i];
     }
     return arraySum;
   }
}
```

```
public class EventLogistics extends FundCalculations {

   protected int maxCapacity;

   public EventLogistics() {
      super();
      maxCapacity = 0;
   }

   . . .
```

```
public class GrantLogistics extends FundCalculations {

   public GrantLogistics(int amountAvailable) {
      super();
      currentBudget = amountAvailable;
   }

   . . .
```

```java
public class EventLogistics extends FundCalculations {

   protected int maxCapacity;

   public EventLogistics() {
      super();
      maxCapacity = 0;
   }

   . . .
```

```java
public class GrantLogistics extends FundCalculations {

   public GrantLogistics(int amountAvailable) {
      super();
      currentBudget = amountAvailable;
   }

   . . .
```

```java
public class EventLogistics {

   . . .

   public void updateWithProfits(int[] siteProfits) {
      for (int i = 0; i < siteProfits.length; i++) {
         currentBudget += siteProfits[i];
      }
      maxCapacity = Math.floor((float) currentBudget * 0.75);
   }
}
```

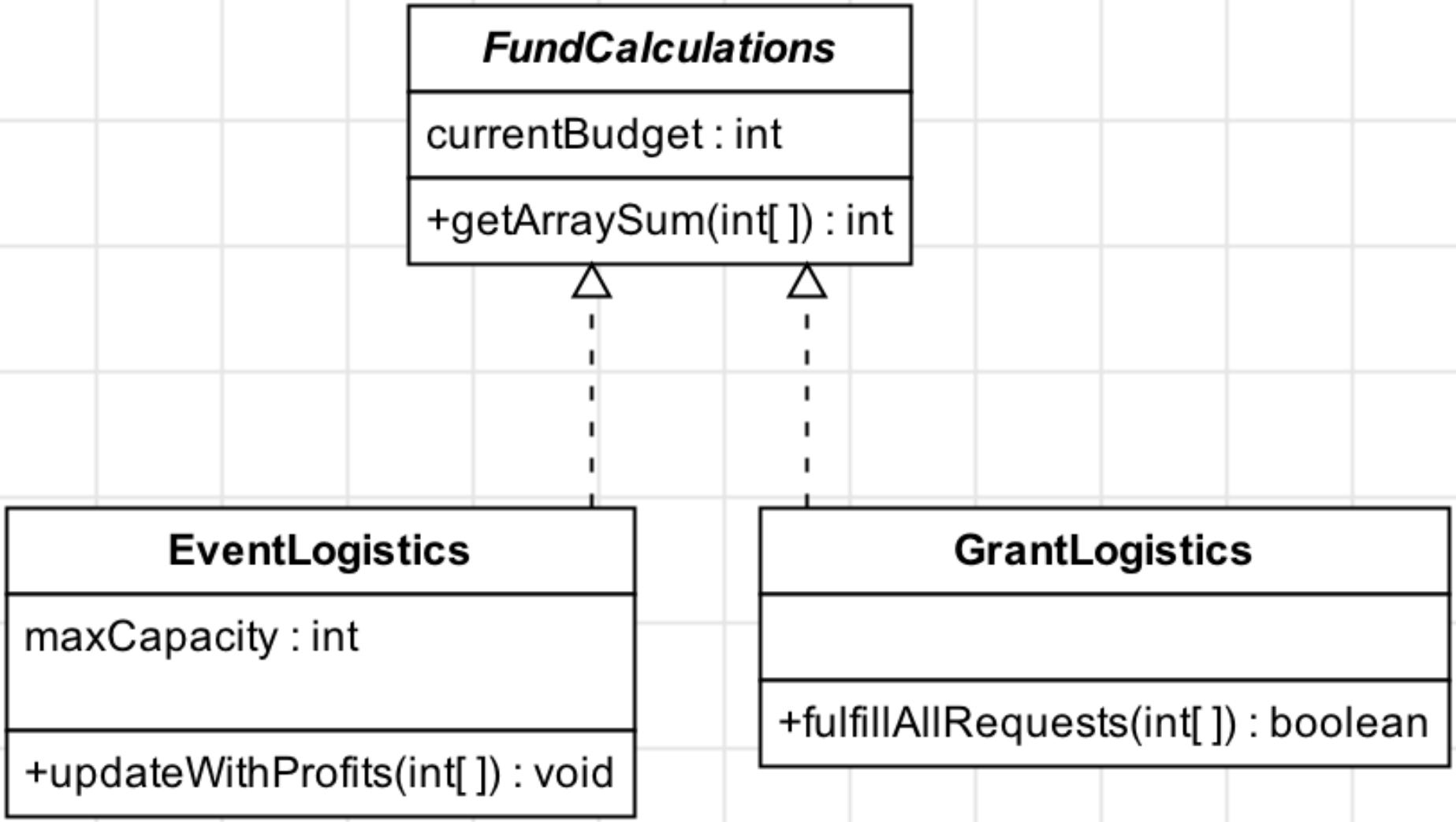
```java
public class GrantLogistics {

   . . .

   public boolean fulfillAllRequests(int[] individualRequests) {
      int totalAmountRequested = 0;
      for (int i = 0; i < individualRequests.length; i++) {
         totalAmountRequested += individualRequests[i];
      }
      return totalAmountRequested <= currentBudget;
   }
}
```

```java
public class EventLogistics {

   . . .

   public void updateWithProfits(int[] siteProfits) {
      for (int i = 0; i < siteProfits.length; i++) {
         currentBudget += siteProfits[i];
      }
      maxCapacity = Math.floor((float) currentBudget * 0.75);
   }
}
```

```java
public class GrantLogistics {

   . . .

   public boolean fulfillAllRequests(int[] individualRequests) {
      int totalAmountRequested = 0;
      for (int i = 0; i < individualRequests.length; i++) {
         totalAmountRequested += individualRequests[i];
      }
      return totalAmountRequested <= currentBudget;
   }
}
```

```java
public class EventLogistics extends FundCalculations {

    . . .

    public void updateWithProfits(int[] siteProfits) {
        currentBudget += getArraySum(siteProfits);


        maxCapacity = Math.floor((float) currentBudget * 0.75);
    }
}
```

```java
public class GrantLogistics extends FundCalculations {

    . . .

    public boolean fulfillAllRequests(int[] individualRequests) {
        int totalAmountRequested = getArraySum(individualRequests);



        return totalAmountRequested <= currentBudget;
    }
}
```


FundCalculations
currentBudget : int
+getArraySum(int[ ]) : int
EventLogistics
maxCapacity : int
+updateWithProfits(int[ ]) : void
GrantLogistics
+fulfillAllRequests(int[ ]) : boolean

Recap: Duplicate Code

- #1 most common code smell
- Hurts understandability, changeability, and testability
- Use **Extract Method** pattern if duplicate code is in same class
- Use **Extract Superclass** if duplicate code is in two different classes

SD2x4.5
Large Class
Chris

Refactoring

- Improve the internal quality (design) of the code
- Simplify future code changes
- Reduce the amount of code to maintain

- Address problems known as **Code Smells**

Code Smell: Large Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Code Smell: Large Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Code Smell: Large Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Code Smell: Large Class

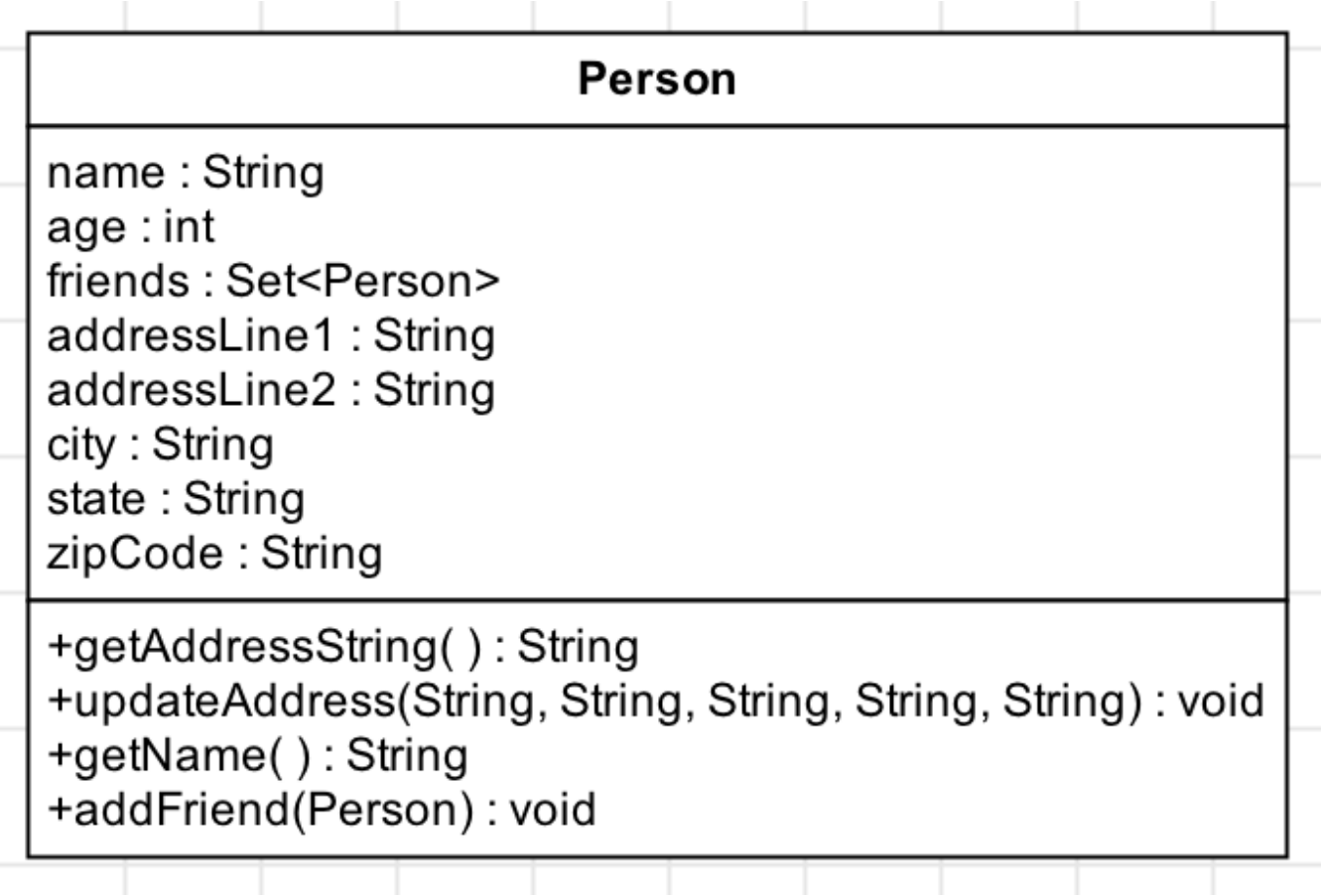
```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

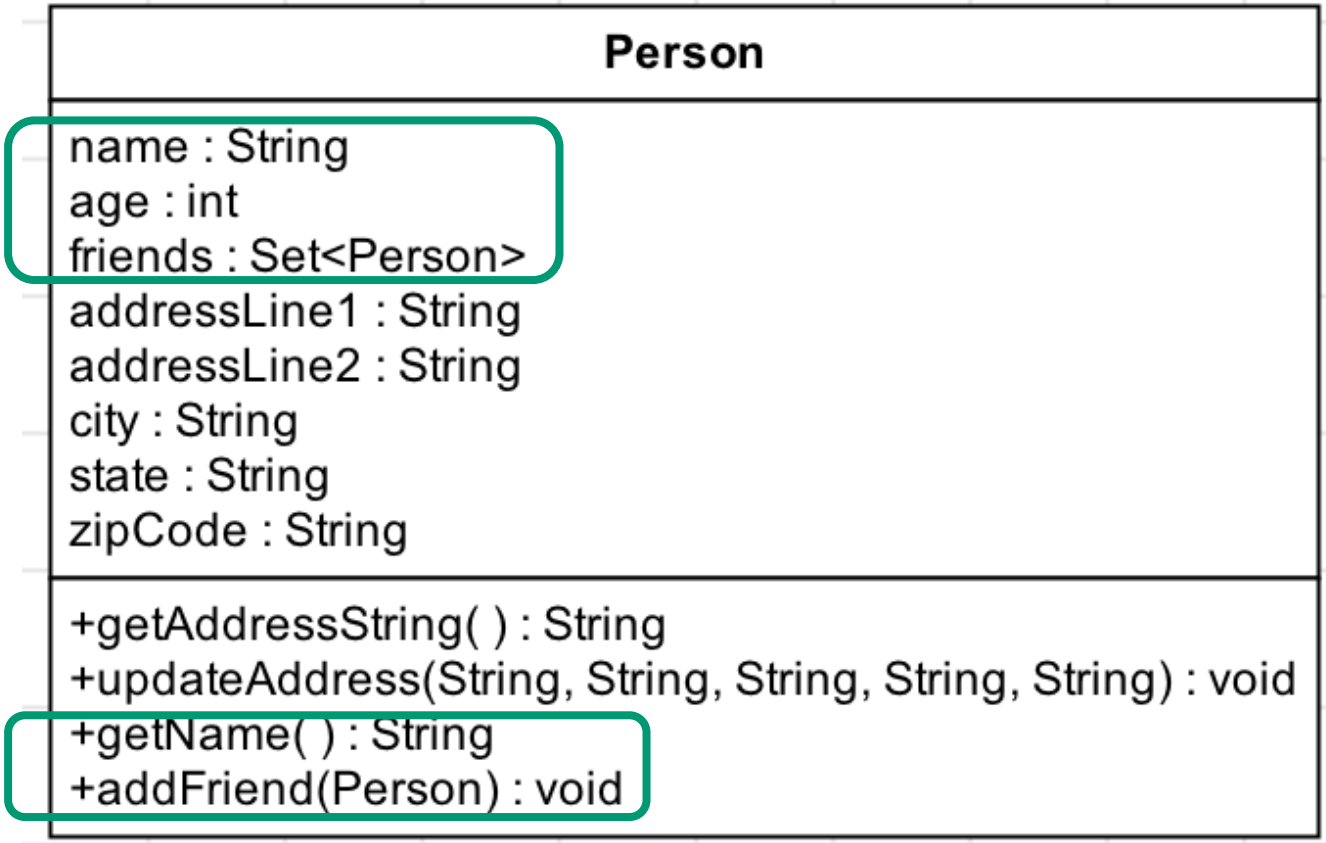
Code Smell: Large Class

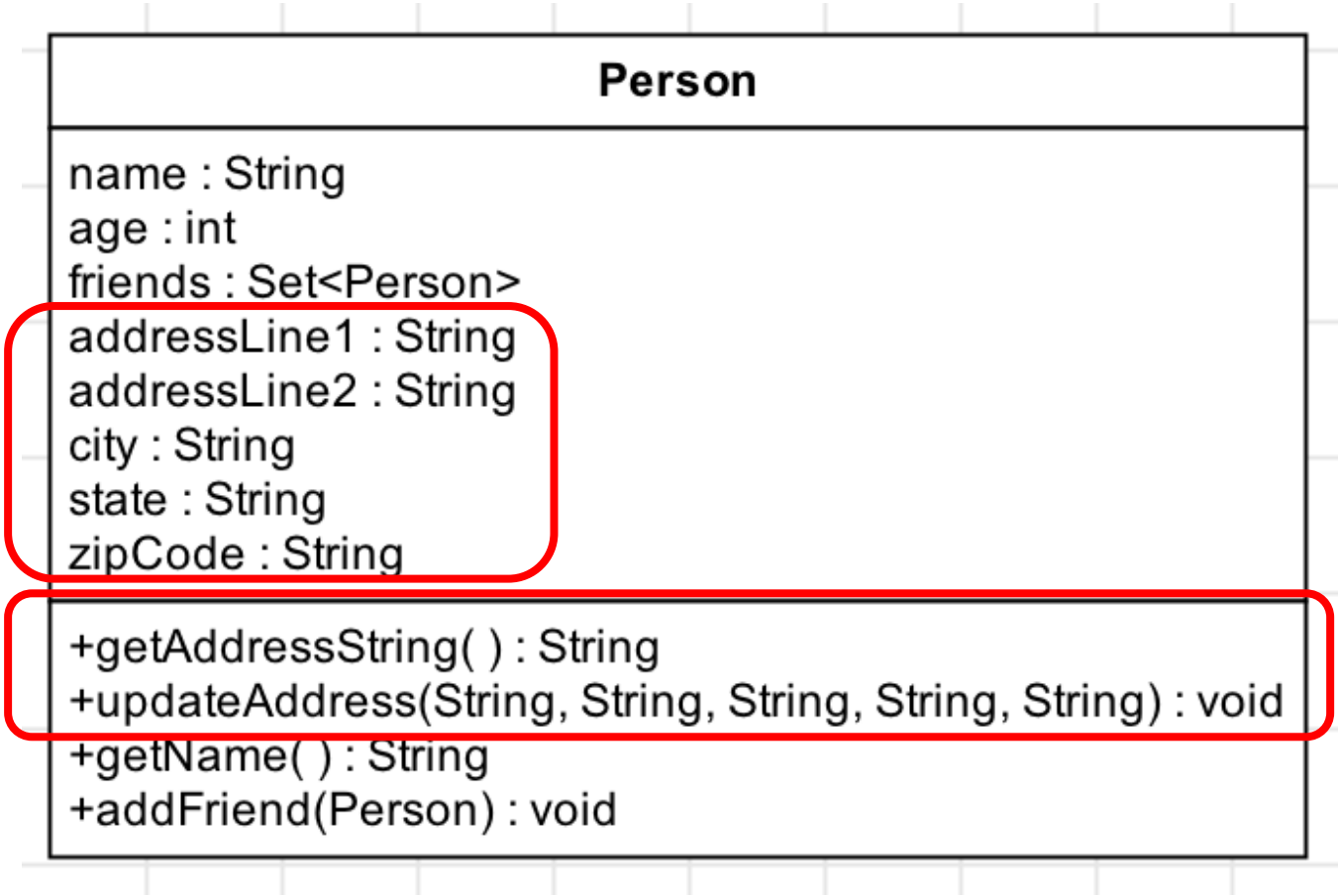
```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Code Smell: Large Class

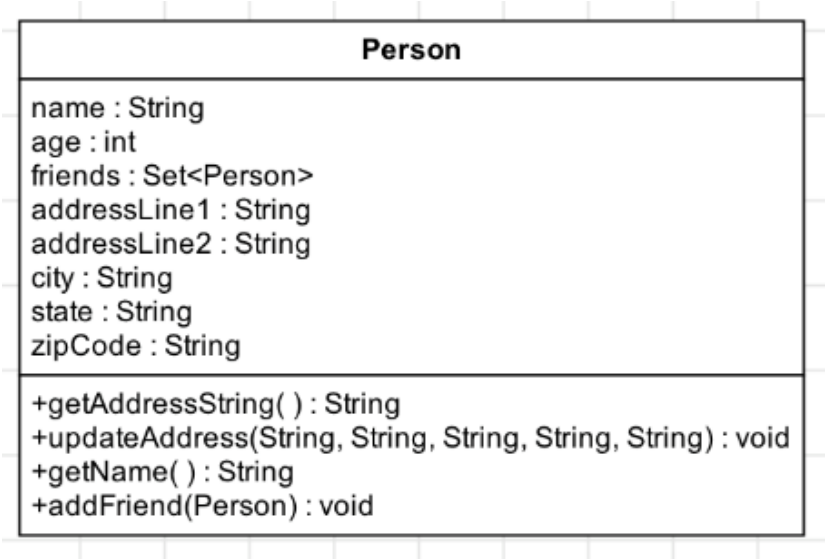
```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

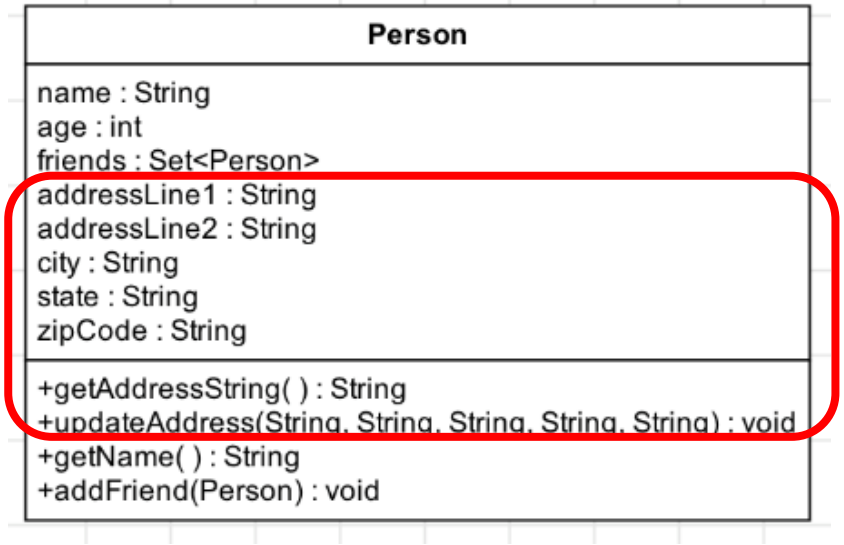

Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void

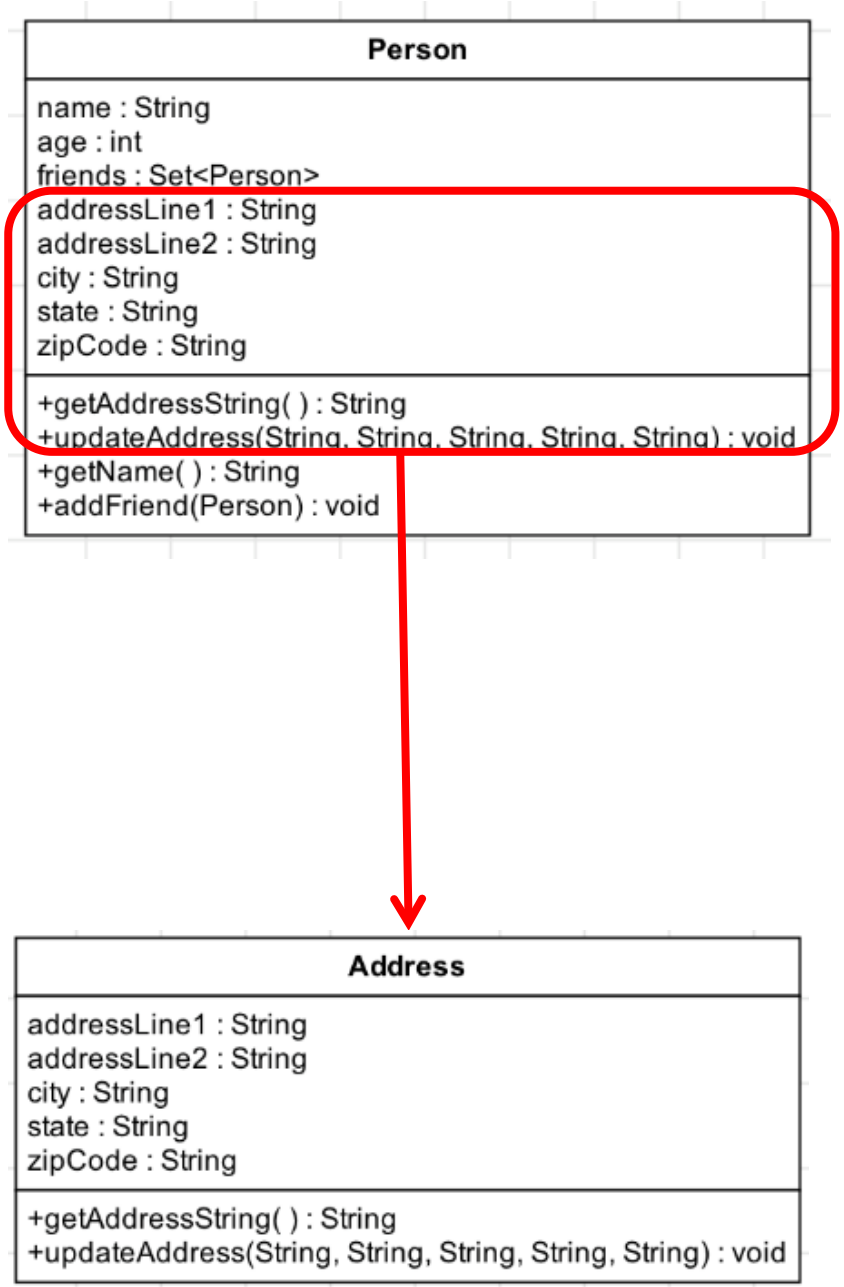

Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void

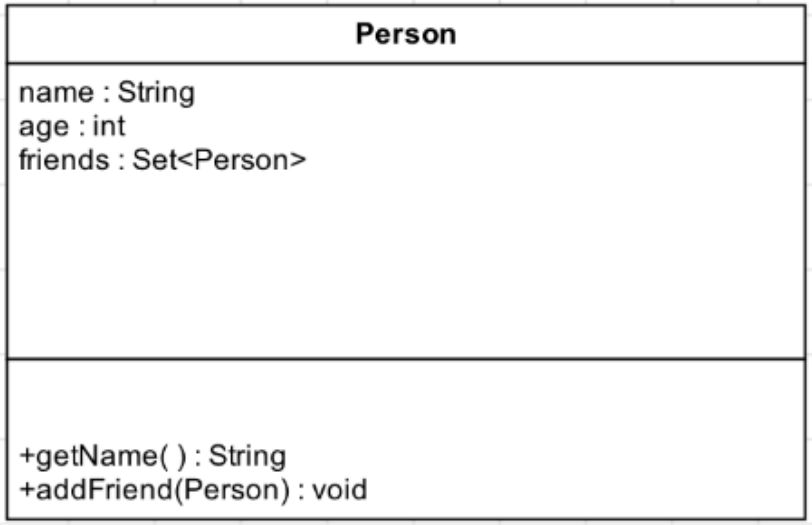

Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void

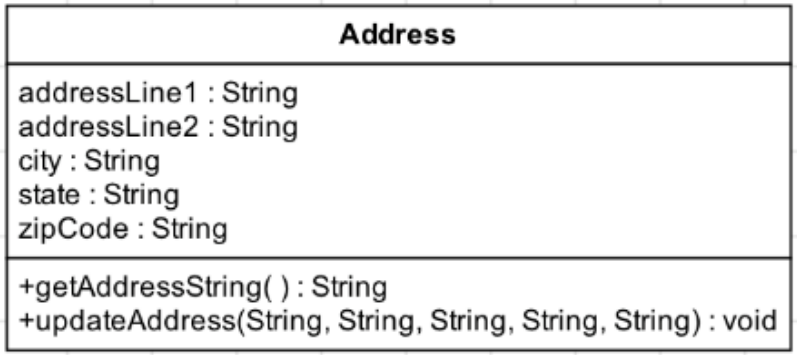
Solution:
Extract Class!

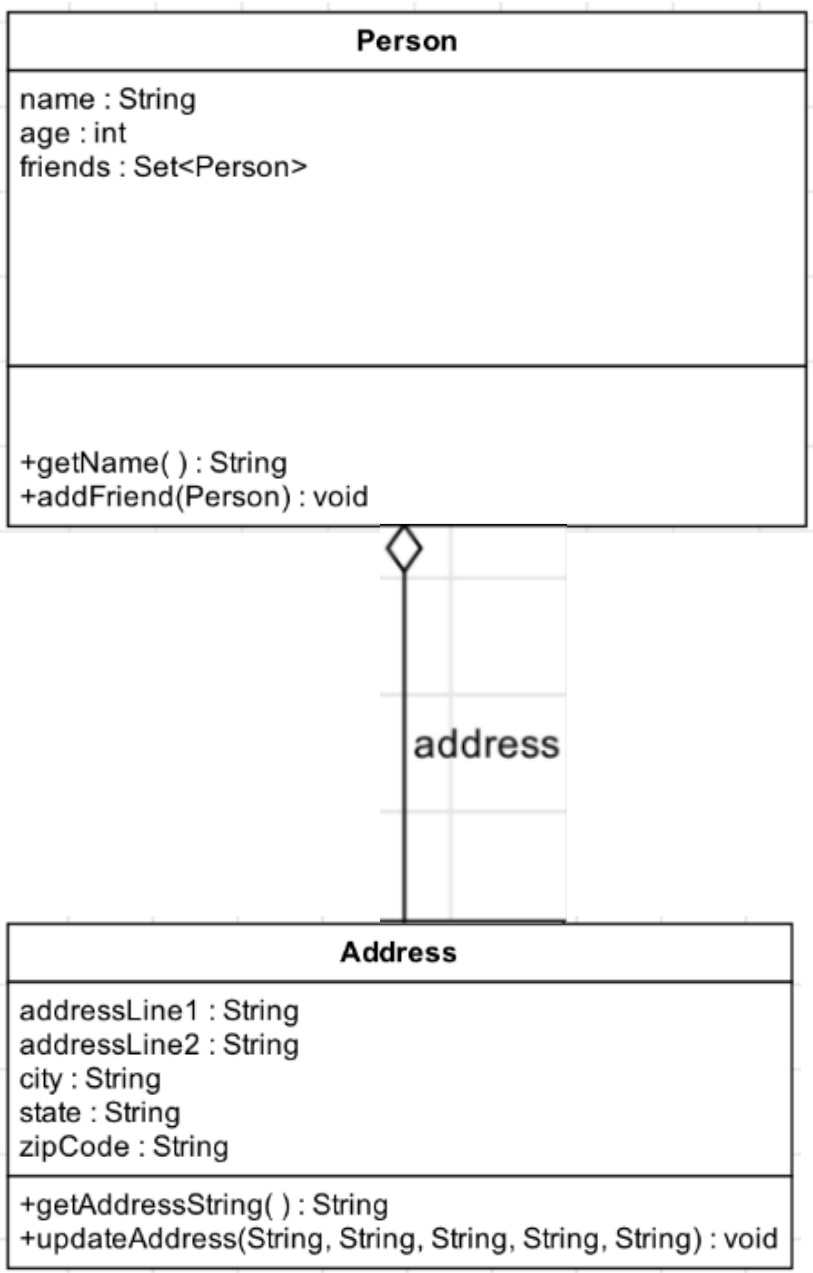

Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void


Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void


Person
name : String
age : int
friends : Set<Person>
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void
+getName( ) : String
+addFriend(Person) : void
Address
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void


Person
name : String
age : int
friends : Set<Person>
+getName( ) : String
+addFriend(Person) : void


Address
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void


Person
name : String
age : int
friends : Set<Person>
+getName( ) : String
+addFriend(Person) : void
address
Address
addressLine1 : String
addressLine2 : String
city : String
state : String
zipCode : String
+getAddressString( ) : String
+updateAddress(String, String, String, String, String) : void

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   String addressLine1;
   String addressLine2;
   String city;
   String state;
   String zipCode;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }





   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }





   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }





   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;





   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() { . . . }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;




   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() {
       return address.getAddressString();
   }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;




   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() {
       return address.getAddressString();
   }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;




   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() {
       return address.getAddressString();
   }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() {
       return address.getAddressString();
   }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) {
     address.updateAddress(addressLine1, addressLine2, city,
           state, zipCode);
   }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Extract Class

```
public class Person {
   String name;
   int age;
   Set<Person> friends;
   Address address;

   public Person(String name, int age,
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) { . . . }

   public String getAddressString() {
       return address.getAddressString();
   }
   public void updateAddress(
       String addressLine1, String addressLine2, String city,
       String state, String zipCode) {
     address.updateAddress(addressLine1, addressLine2, city,
           state, zipCode);
   }

   public void getName() { . . . }
   public void addFriend(Person p) { . . . }
}
```

Here’s another example!

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . .
}
```

Code Smell: Large Class

```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) { . . . }
   . . . keep going! -->
}
```

```
public class Student {
   . . .

   public void completedCourse(String course,
           double courseGradePoint) {
      coursesTaken.put(course, courseGradePoint);
      updateGradePointAverage();
   }
   private void updateGradePointAverage() { . . . }

   // getters and setters specific to student information
   public String getStudentID() { . . . }
   public int getGraduationYear() { . . . }
   public double getGradePointAverage() { . . . }
   . . .

   // getters and setters for basic information—that is,
   // people in general, not just students, have this information
   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .
}
```

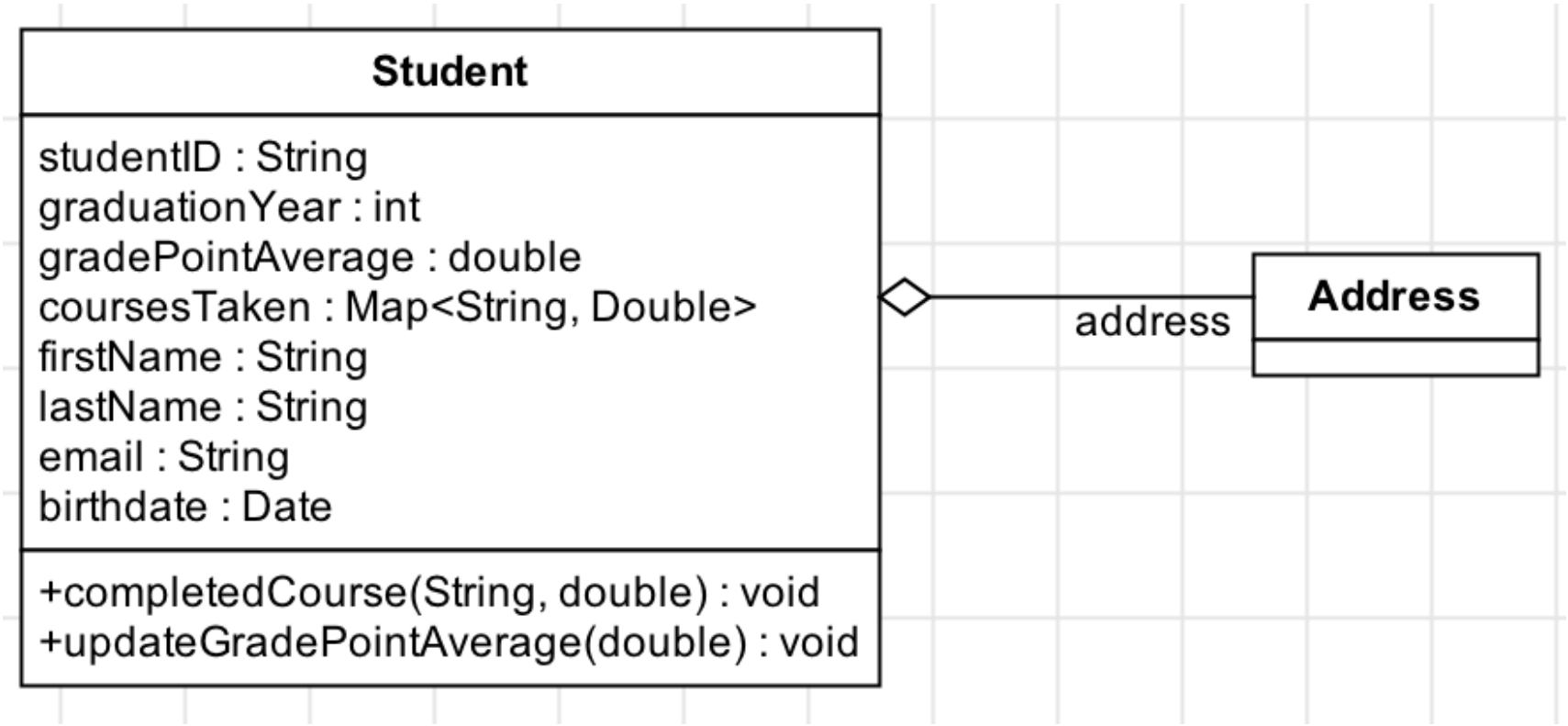
```java
public class Student {
   . . .

   public void completedCourse(String course,
           double courseGradePoint) {
      coursesTaken.put(course, courseGradePoint);
      updateGradePointAverage();
   }
   private void updateGradePointAverage() { . . . }

   // getters and setters specific to student information
   public String getStudentID() { . . . }
   public int getGraduationYear() { . . . }
   public double getGradePointAverage() { . . . }
   . . .

   // getters and setters for basic information—that is,
   // people in general, not just students, have this information
   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .
}
```

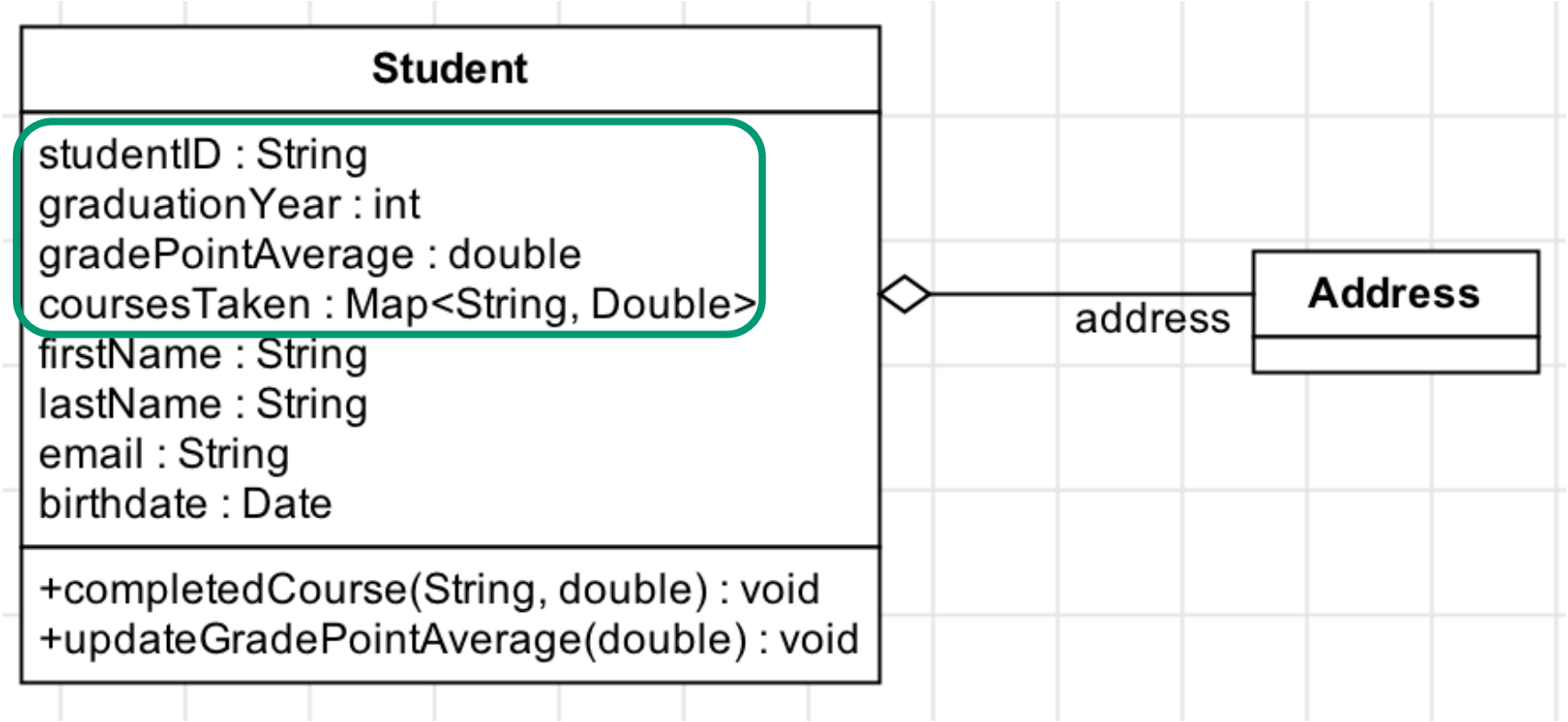
```java
public class Student {
   . . .

   public void completedCourse(String course,
           double courseGradePoint) {
      coursesTaken.put(course, courseGradePoint);
      updateGradePointAverage();
   }
   private void updateGradePointAverage() { . . . }

   // getters and setters specific to student information
   public String getStudentID() { . . . }
   public int getGraduationYear() { . . . }
   public double getGradePointAverage() { . . . }
   . . .

   // getters and setters for basic information—that is,
   // people in general, not just students, have this information
   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .
}
```

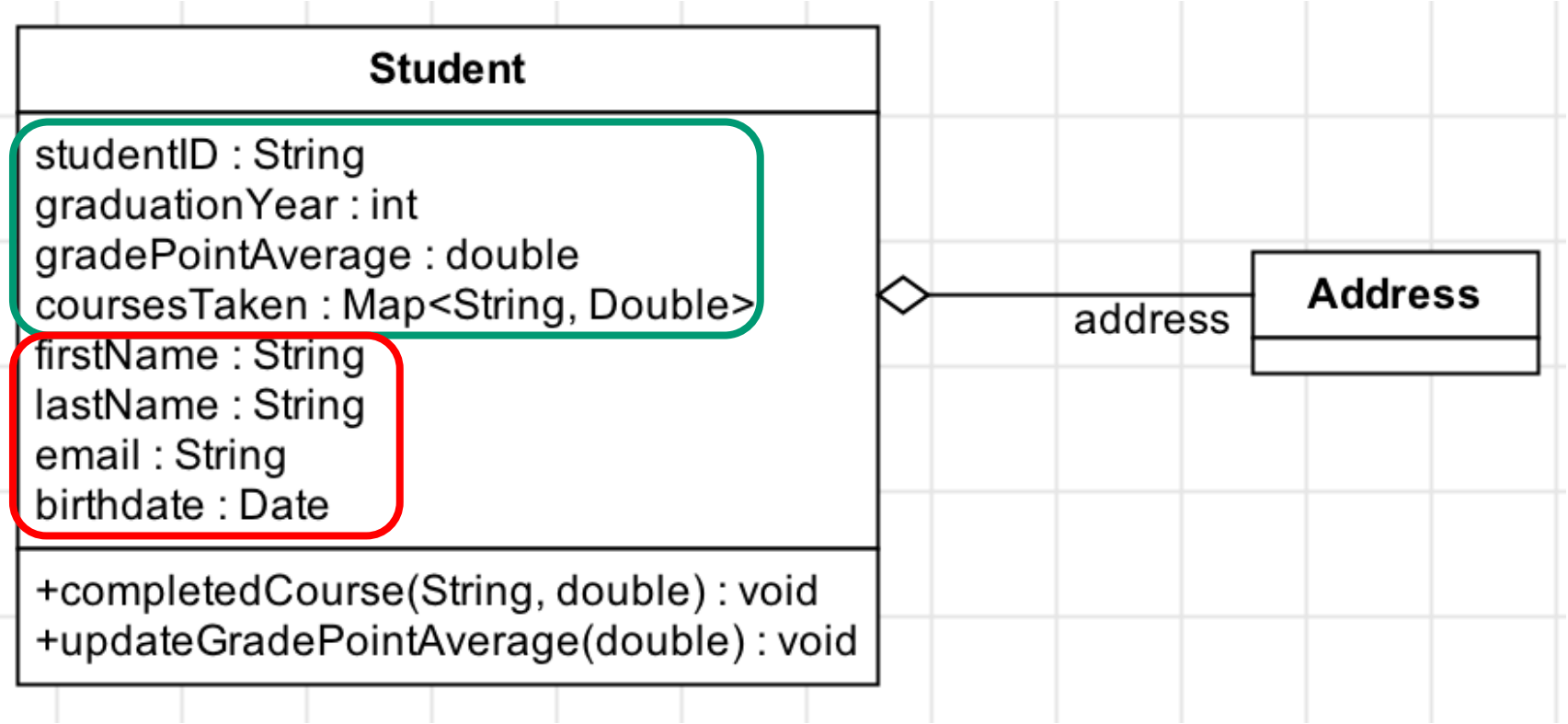
```java
public class Student {
   . . .

   public void completedCourse(String course,
           double courseGradePoint) {
      coursesTaken.put(course, courseGradePoint);
      updateGradePointAverage();
   }
   private void updateGradePointAverage() { . . . }

   // getters and setters specific to student information
   public String getStudentID() { . . . }
   public int getGraduationYear() { . . . }
   public double getGradePointAverage() { . . . }
   . . .

   // getters and setters for basic information—that is,
   // people in general, not just students, have this information
   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .
}
```

```java
public class Student {
   . . .

   public void completedCourse(String course,
           double courseGradePoint) {
      coursesTaken.put(course, courseGradePoint);
      updateGradePointAverage();
   }
   private void updateGradePointAverage() { . . . }

   // getters and setters specific to student information
   public String getStudentID() { . . . }
   public int getGraduationYear() { . . . }
   public double getGradePointAverage() { . . . }
   . . .

   // getters and setters for basic information—that is,
   // people in general, not just students, have this information
   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .
}
```

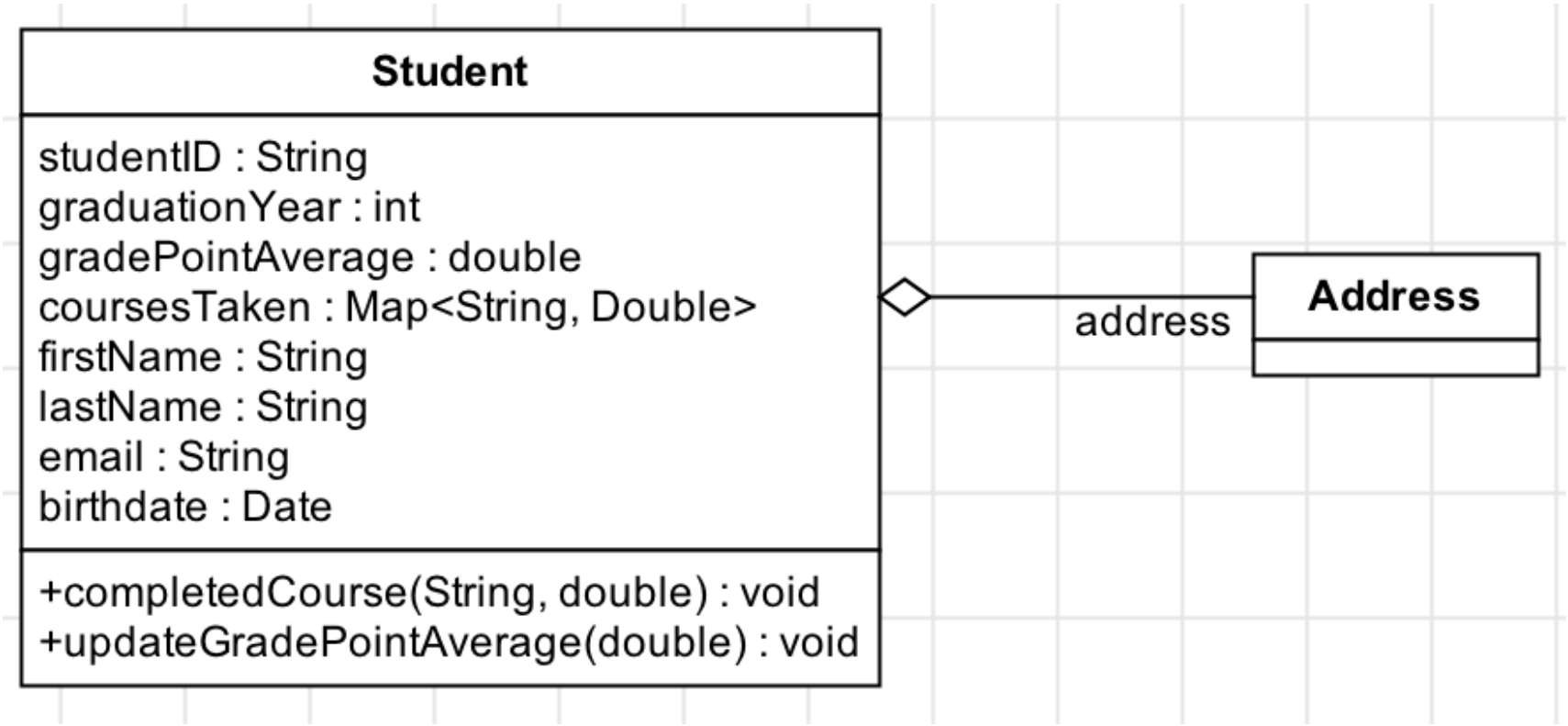
```
public class Student {
   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduateYear) { . . . }
   . . .
}
```

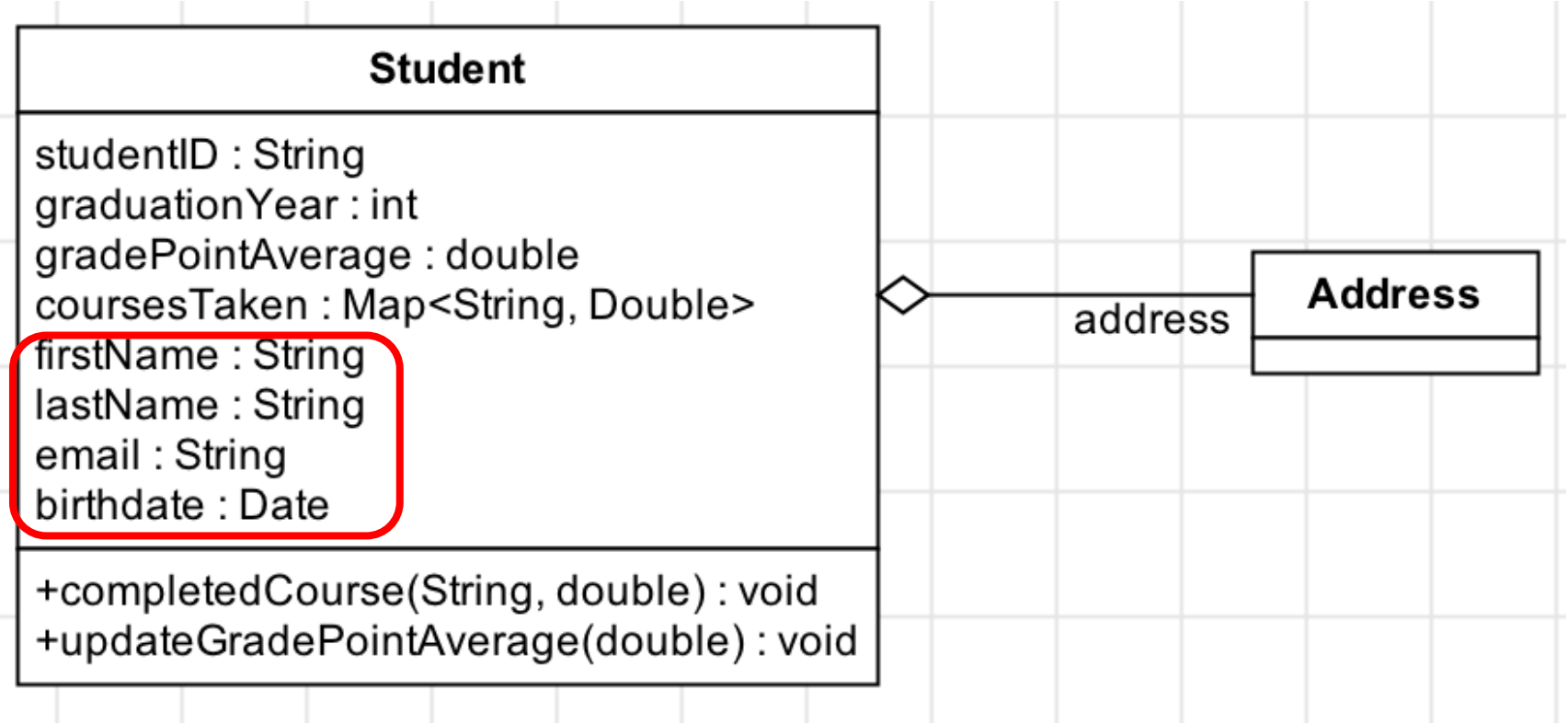

Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
firstName : String
lastName : String
email : String
birthdate : Date
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
address
Address

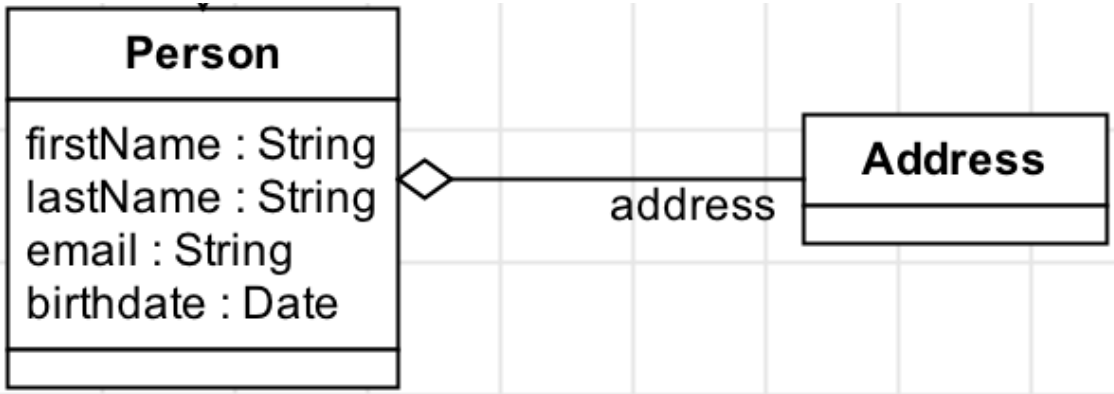

Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
firstName : String
lastName : String
email : String
birthdate : Date
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
address
Address

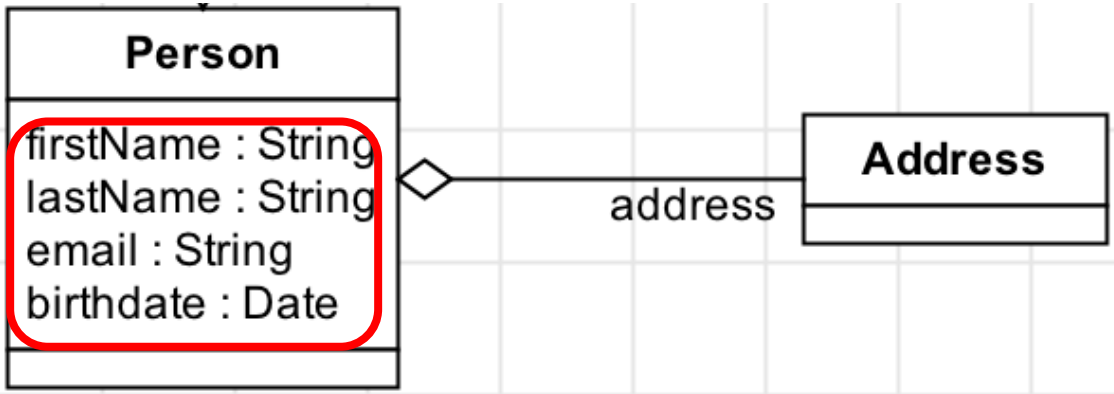

Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
firstName : String
lastName : String
email : String
birthdate : Date
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
address
Address

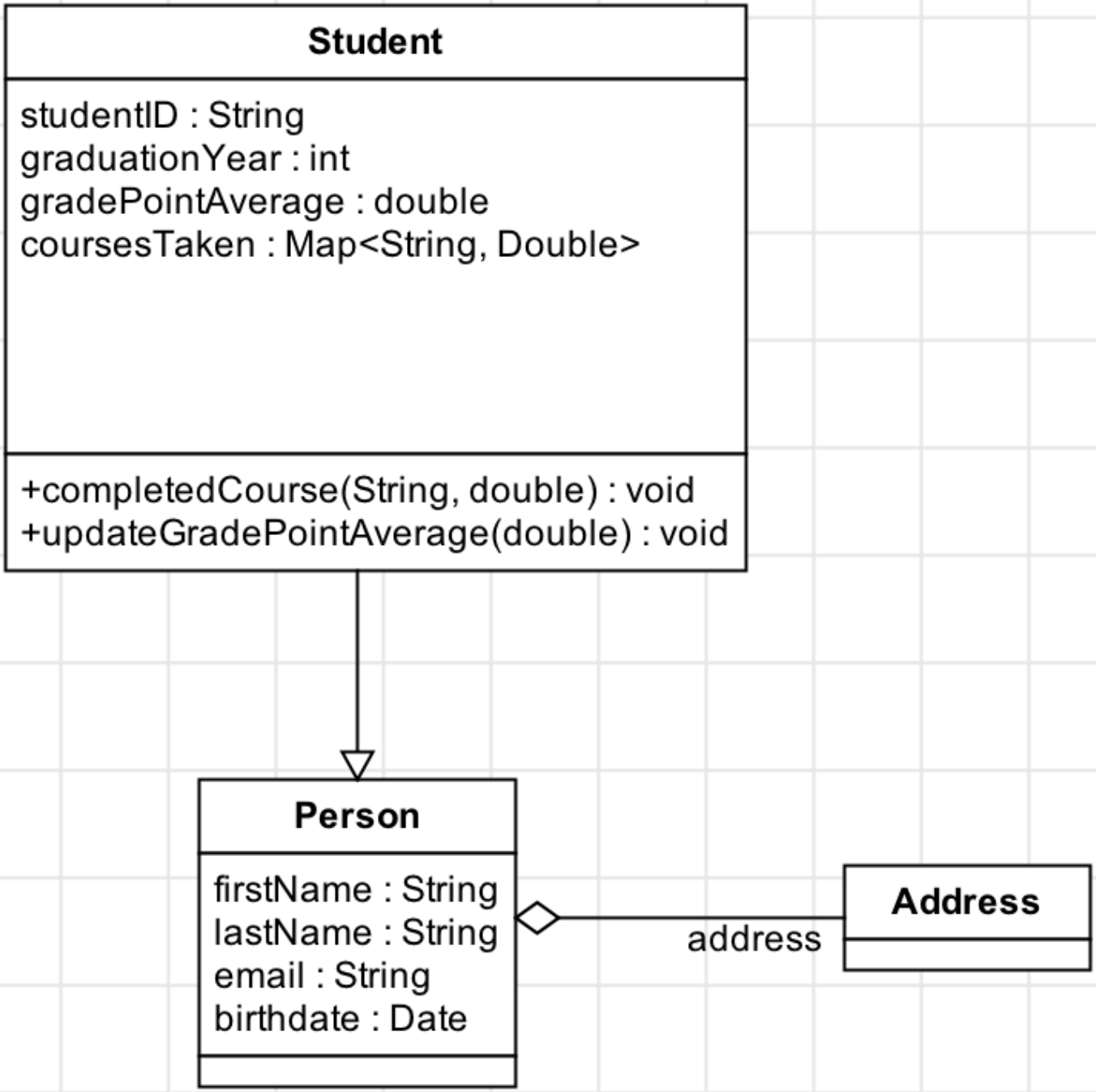
Solution:
Extract Superclass!


Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
firstName : String
lastName : String
email : String
birthdate : Date
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
address
Address


Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
firstName : String
lastName : String
email : String
birthdate : Date
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
address
Address


Person
firstName : String
lastName : String
email : String
birthdate : Date
address
Address


Person
firstName : String
lastName : String
email : String
birthdate : Date
address
Address


Student
studentID : String
graduationYear : int
gradePointAverage : double
coursesTaken : Map<String, Double>
+completedCourse(String, double) : void
+updateGradePointAverage(double) : void
Person
firstName : String
lastName : String
email : String
birthdate : Date
address
Address

Extract Superclass

```
public class Person {

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Person(String firstName, String lastName, String email,
       Address address, Date birthdate) { . . . }

   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .

}
```

Extract Superclass

```
public class Person {

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Person(String firstName, String lastName, String email,
       Address address, Date birthdate) { . . . }

   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .

}
```

Extract Superclass

```
public class Person {

   private String firstName;
   private String lastName;
   private String email;
   private Address address;
   private Date birthdate;

   public Person(String firstName, String lastName, String email,
       Address address, Date birthdate) { . . . }

   public String getFullName() { . . . }
   public String getFirstName() { . . . }
   public String getLastName() { . . . }
   public String getEmail() { . . . }
   public void updateEmail(String newEmail) { . . . }
   . . .

}
```

Extract Superclass

```
public class Student extends Person {

   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) {
      super(firstName, lastName, email, address, birthdate);
      this.studentID = studentID;
      this.graduationYear = graduationYear;
   }
   . . .
}
```

Extract Superclass

```
public class Student extends Person {

   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) {
      super(firstName, lastName, email, address, birthdate);
      this.studentID = studentID;
      this.graduationYear = graduationYear;
   }
   . . .
}
```

Extract Superclass

```
public class Student extends Person {

   private String studentID;
   private int graduationYear;
   private double gradePointAverage = 4.0;
   // (String) course name, (Double) grade point
   private Map<String, Double> coursesTaken =
   new HashMap<String, Double>();

   public Student(String firstName, String lastName,
         String email, Address address, Date birthdate,
         String studentID, int graduationYear) {
      super(firstName, lastName, email, address, birthdate);
      this.studentID = studentID;
      this.graduationYear = graduationYear;
   }
   . . .
}
```

Recap: Large Class

- Hurts understandability, changeability, and testability
- Use **Extract Class** pattern to create a “has-a” relationship with new class
- Use **Extract Superclass** pattern to create an “is-a” relationship with new class

SD2x4.6

Reliability

Chris

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

Reliability:

Probability of failure-free operation in a specific environment over a specific period of time

Reliability:

Likelihood of producing correct output/behavior.

How do we make software more reliable?

- **Fault prevention:** make sure there are no bugs in the first place
- **Fault removal:** find and fix bugs (testing and debugging)
- **Fault tolerance:** avoid failure even in the presence of faults

Defensive programming

- We don’t always know how other people will use our code

- Need to protect against mistakes made by:
  - people who use your code
  - people whose code you use
  - yourself!

What if you notice something wrong?

- **Do nothing:** “garbage in, garbage out”
- **Notify caller:** “garbage in, error out”
- **Halt:** “garbage in, nothing out”

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

Example: BankAccount

```
public class BankAccount {

    private double balance = 0;

    public BankAccount(double initBalance) {
        balance = initBalance;
    }

    public double getBalance() { return balance; }

    public void withdraw(double amount) { . . . }

    public void deposit(double amount) { . . . }

    public void transfer(BankAccount other, double amount) {
        this.deposit(amount);
        other.withdraw(amount);
    }
}
```

How can we notify the caller?

- Error Codes
- Unchecked Exceptions
- Checked Exceptions
- Assertion Violations

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

1. Notify Caller: Error Codes

```
public static final int NULL_ARGUMENT = -1;
public static final int NEGATIVE_BALANCE = -2;
public static final int SUCCESS = 0;

public int transfer(BankAccount other, double amount) {
    if (other == null)
        return BankAccount.NULL_ARGUMENT;
    else if (this.balance < 0)
        return BankAccount.NEGATIVE_BALANCE;
    this.deposit(amount);
    other.withdraw(amount);
    return BankAccount.SUCCESS;
}
```

2. Notify Caller: Unchecked Exceptions

```
public void transfer(BankAccount other, double amount) {
   if (other == null)
       throw new IllegalArgumentException("BankAccount is null");
   else if (this.balance < 0)
       throw new IllegalStateException("Balance is negative");
   this.deposit(amount);
   other.withdraw(amount);
}
```

2. Notify Caller: Unchecked Exceptions

```
public void transfer(BankAccount other, double amount) {
   if (other == null)
       throw new IllegalArgumentException("BankAccount is null");
   else if (this.balance < 0)
       throw new IllegalStateException("Balance is negative");
   this.deposit(amount);
   other.withdraw(amount);
}
```

2. Notify Caller: Unchecked Exceptions

```
public void transfer(BankAccount other, double amount) {
   if (other == null)
       throw new IllegalArgumentException("BankAccount is null");
   else if (this.balance < 0)
       throw new IllegalStateException("Balance is negative");
   this.deposit(amount);
   other.withdraw(amount);
}
```

2. Notify Caller: Unchecked Exceptions

```
public void transfer(BankAccount other, double amount) {
   if (other == null)
       throw new IllegalArgumentException("BankAccount is null");
   else if (this.balance < 0)
       throw new IllegalStateException("Balance is negative");
   this.deposit(amount);
   other.withdraw(amount);
}
```

2. Notify Caller: Unchecked Exceptions

```
public void transfer(BankAccount other, double amount) {
   if (other == null)
       throw new IllegalArgumentException("BankAccount is null");
   else if (this.balance < 0)
       throw new IllegalStateException("Balance is negative");
   this.deposit(amount);
   other.withdraw(amount);
}
```

3. Notify Caller: Checked Exceptions

```
public void transfer(BankAccount other, double amount)
    throws NullArgumentException, NegativeBalanceException {
    if (other == null)
        throw new NullArgumentException();
    else if (this.balance < 0)
        throw new NegativeBalanceException();
    this.deposit(amount);
    other.withdraw(amount);
}
```

3. Notify Caller: Checked Exceptions

```
public void transfer(BankAccount other, double amount)
    throws NullArgumentException, NegativeBalanceException {
    if (other == null)
        throw new NullArgumentException();
    else if (this.balance < 0)
        throw new NegativeBalanceException();
    this.deposit(amount);
    other.withdraw(amount);
}
```

3. Notify Caller: Checked Exceptions

```
public void transfer(BankAccount other, double amount)
    throws NullArgumentException, NegativeBalanceException {
    if (other == null)
        throw new NullArgumentException();
    else if (this.balance < 0)
        throw new NegativeBalanceException();
    this.deposit(amount);
    other.withdraw(amount);
}
```

3. Notify Caller: Checked Exceptions

```
public void transfer(BankAccount other, double amount)
    throws NullArgumentException, NegativeBalanceException {
    if (other == null)
        throw new NullArgumentException();
    else if (this.balance < 0)
        throw new NegativeBalanceException();
    this.deposit(amount);
    other.withdraw(amount);
}
```

3. Notify Caller: Checked Exceptions

```
public void transfer(BankAccount other, double amount)
    throws NullArgumentException, NegativeBalanceException {
    if (other == null)
        throw new NullArgumentException();
    else if (this.balance < 0)
        throw new NegativeBalanceException();
    this.deposit(amount);
    other.withdraw(amount);
}
```

4. Notify Caller: Assertion Violations

```
public void transfer(BankAccount other, double amount) {
    assert other != null;
    assert this.balance >= 0;
    this.deposit(amount);
    other.withdraw(amount);
}
```

4. Notify Caller: Assertion Violations

```
public void transfer(BankAccount other, double amount) {
    assert other != null;
    assert this.balance >= 0;
    this.deposit(amount);
    other.withdraw(amount);
}
```

4. Notify Caller: Assertion Violations

```
public void transfer(BankAccount other, double amount) {
    assert other != null;
    assert this.balance >= 0;
    this.deposit(amount);
    other.withdraw(amount);
}
```

4. Notify Caller: Assertion Violations

```
public void transfer(BankAccount other, double amount) {
    assert other != null;
    assert this.balance >= 0;
    this.deposit(amount);
    other.withdraw(amount);
}
```

Exceptions vs. Assertions

- Exceptions are for things that should not happen (but could be anticipated)

- Assertions are for things that should **NEVER** happen (and indicate a bug)

Notify Caller: Assertions

```
public void transfer(BankAccount other, double amount) {
    if (other == null) { . . . }
    if (this.balance < 0) { . . . }
    if (amount < 0) { . . . }
    this.deposit(amount);
    other.withdraw(amount);
    assert this.balance >= 0;
}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    for (String name : names) {
         if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    for (String name : names) {
         if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
         if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
         if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
         if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
        if (name == null) continue;
        if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
        if (name == null) continue;
        if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
        if (name == null) continue;
        if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
        if (name == null) continue;
        if (name.size() == 0) continue;
        if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Defensive Programming: Trust No One!

```
public int countMatches (char letter) {

    int count = 0;

    List<String> names = dataSource.getAllNames();

    if (names == null) throw new IllegalStateException();

    for (String name : names) {
        if (name == null) continue;
        if (name.size() == 0) continue;
        if (name.charAt(0) == letter)
            count++;
    }

    return count;

}
```

Recap: Defensive programming

- Protect against mistakes made by:
  - people who use your code
  - people whose code you use
  - yourself!

- Consider the mechanisms for notifying the caller that something has gone wrong

SD2x4.7

Efficiency - 1

Chris

How do we maintain software?

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space

Software Quality is about **tradeoffs**

If you make your code faster, what could get worse?

- Understandability, Internal Quality, etc.
- Correctness/Functionality
- Reliability
- Security
- Memory usage
- Code size
- Development effort
- Compilation time
- …

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs;

these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered.

We *should* forget about small efficiencies, say, about 97% of the time;

premature optimization is the root of all evil.

-- Donald Knuth, 1974

“Yet we should not pass up our opportunities in that critical 3%.”

Rules of Thumb!

- Rule #1: Use the right **algorithm** or **data structure**
- ***No, seriously, that's the most important thing***

- Make the common case fast (80/20 rule)
- *Measure! Don't guess!*

- Avoid unnecessary work
- *Efficient means "not inefficient"*

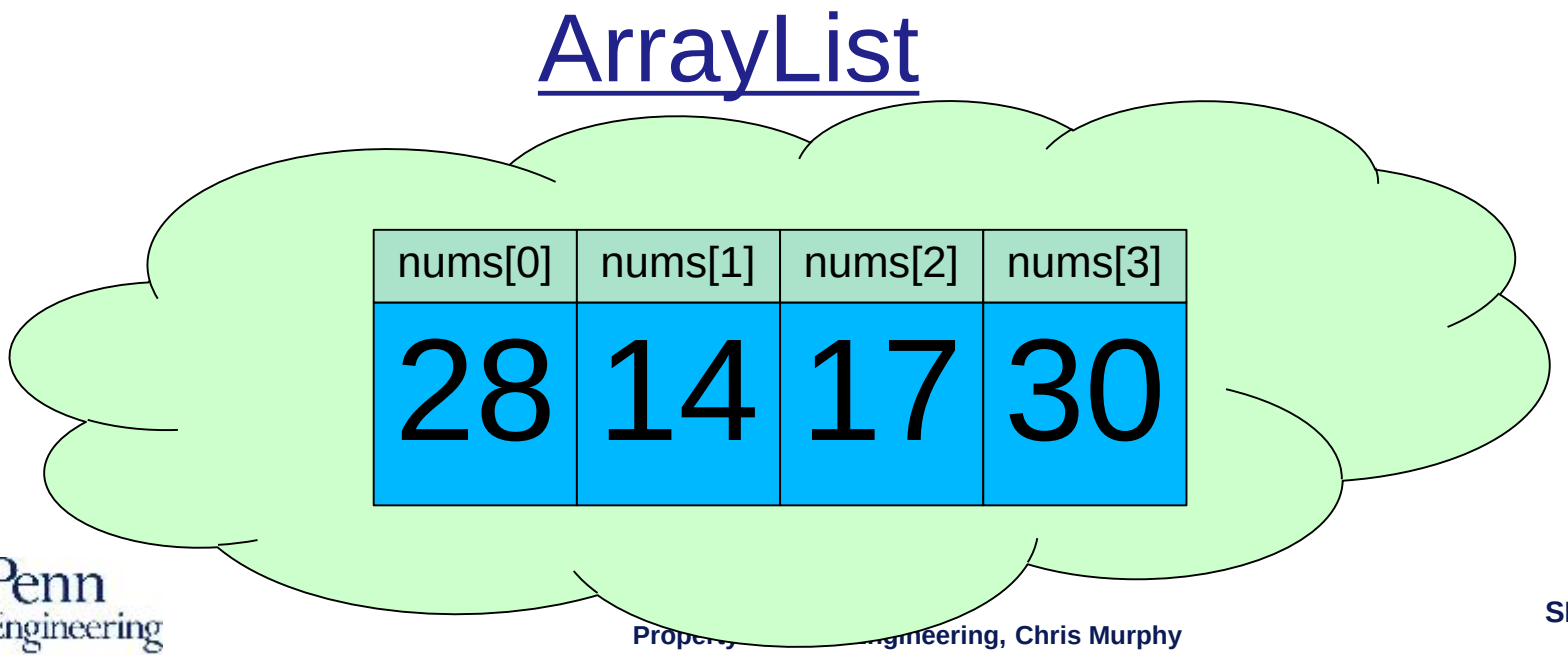
- Use micro-optimizations wisely

Review: Data Structures in Java

- **Lists**
  - ordered collection of elements
  - allow for duplicates

- **Sets**
  - (potentially) unordered collection of elements
  - do not allow for duplicates

head
tail
LinkedList
28
14
17
30
null


ArrayList
nums[0]
nums[1]
nums[2]
nums[3]
28
14
17
30

Review: Lists in Java

- **LinkedList:** collection of linked nodes
- **ArrayList:** object that wraps underlying array

	LinkedList	**ArrayList**
Adding an element	O(1) to front or rear	“amortized” O(1)
Removing an element	O(1) from front or rear	O(1) from rear O(n) otherwise
Index-based access	O(n)	O(1)
Finding an element	O(n)	O(n)

Review: Lists in Java

- **LinkedList:** collection of linked nodes
- **ArrayList:** object that wraps underlying array

	LinkedList	**ArrayList**
Adding an element	O(1) to front or rear	"amortized" O(1)
Removing an element	O(1) from front or rear	O(1) from rear O(n) otherwise
Index-based access	O(n)	O(1)
Finding an element	O(n)	O(n)

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

```
public int elementsInCommon(List list1, List list2) {
    int count = 0;
    for (int i = 0; i < list1.size(); i++) {
        for (int j = 0; j < list2.size(); j++) {
            if (list1.get(i).equals(list2.get(j))) {
                count++;
            }
        }
    }
}
```

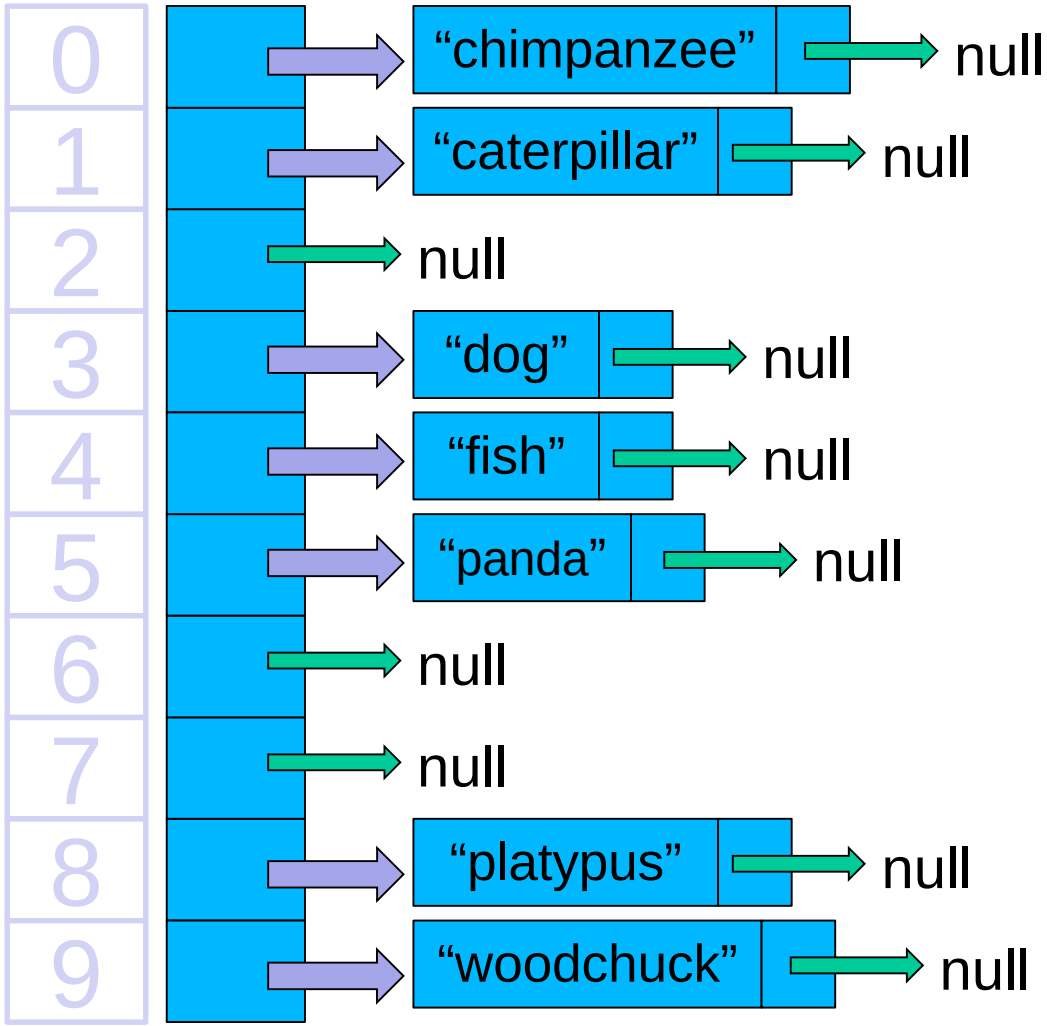
list1: 2,500 distinct elements

list2: 17,000 distinct elements

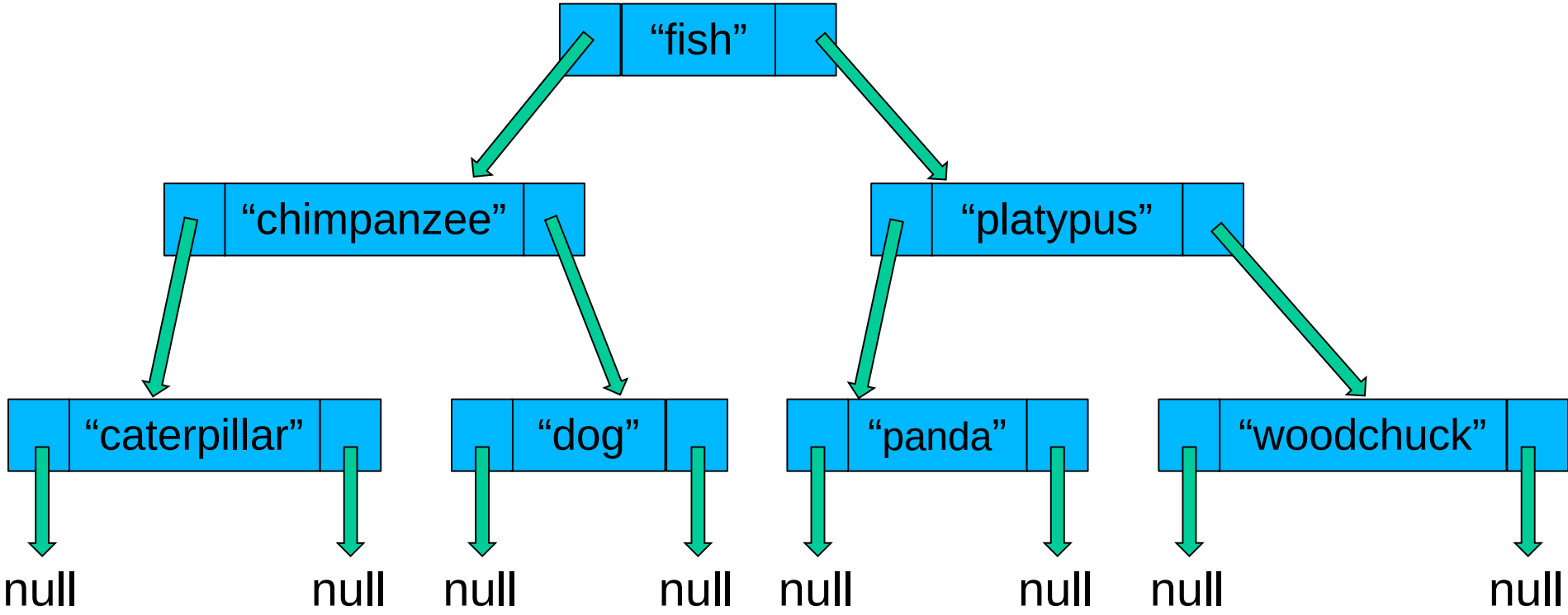
ArrayList: 5 seconds

LinkedList: over 18 minutes!

HashSet


0
1
2
3
4
5
6
7
8
9
"chimpanzee"
null
"caterpillar"
null
null
"dog"
null
"fish"
null
"panda"
null
null
null
"platypus"
null
"woodchuck"
null

TreeSet


"fish"
"chimpanzee"
"platypus"
"caterpillar"
"dog"
"panda"
"woodchuck"
null
null
null
null
null
null
null
null

Review: Sets in Java

- **HashSet:** array of LinkedLists
- **TreeSet:** self-balancing binary search tree

	HashSet	**TreeSet**
Adding an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$
Removing an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$
Finding an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$

Review: Sets in Java

- **HashSet:** array of LinkedLists
- **TreeSet:** self-balancing binary search tree

	HashSet	TreeSet
Adding an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$
Removing an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$
Finding an element	O(1) in best case O(n) in worst case	$O(\log_2 n)$

- **Remember!** TreeSets maintain the elements' "natural order" but HashSets do not

Measuring Execution Time

```
long start = System.currentTimeMillis();

myMethod();

long end = System.currentTimeMillis();

long myMethodExecutionTime = end – start;
```

Measuring Execution Time

```
long start = System.currentTimeMillis();

myMethod();

long end = System.currentTimeMillis();

long myMethodExecutionTime = end – start;
```

Measuring Execution Time

```
long start = System.currentTimeMillis();

myMethod();

long end = System.currentTimeMillis();

long myMethodExecutionTime = end – start;
```

Measuring Execution Time

```
long start = System.currentTimeMillis();

myMethod();

long end = System.currentTimeMillis();

long myMethodExecutionTime = end – start;
```

Measuring Execution Time

```
long start = System.currentTimeMillis();

myMethod();

long end = System.currentTimeMillis();

long myMethodExecutionTime = end – start;
```

Limitations of “Wall Clock Time”

- This approach measures the actual elapsed time (e.g. as measured by a “wall clock”) and not **only** the time executing your code

- This can include:
  - Time the computer spent running other processes
  - Time spent loading class definitions into the JVM

- Better approach: use profiling tools such as Java Heap Analysis Tool (jhat)

Recap: Efficiency

- Improving execution time is a **tradeoff**: other aspects of quality may get worse
- **Rule #1:** always use the right algorithm or data structure
- Make the common case fast
- Measure to check that your code is actually faster

SD2x4.8

Efficiency – 2

Chris

Efficiency:

"Not inefficient"

"Don't do unnecessary work"

How can we make this more efficient?

```
public boolean imSlow(String text) {

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (NullPointerException e) {
      return false;
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (NullPointerException e) {
      return false;
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (NullPointerException e) {
      return false;
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   try {
      if (text.charAt(0) == '\n') {
         return true;
      }
   }
   catch (StringIndexOutOfBoundsException e) {
      return false;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

1. Don’t use exceptions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("") || text.length() == 0) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("")) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.equals("")) {
      return false;
   }
   else if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }




   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
      return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
       return true;
   }

   return false;
}
```

2. Don’t check conditions unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
       return true;
   }

   return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
       return true;
   }

   return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
       return true;
   }

   return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (text.charAt(0) == '\n') {
       return true;
   }

   if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
       return true;
   }

   return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
    if (text == null) {
        return false;
    }

    if (text.length() == 0) {
        return false;
    }

    char firstChar = text.charAt(0);

    if (text.charAt(0) == '\n') {
        return true;
    }

    if (text.charAt(0) >= '0' && text.charAt(0) <= '9') {
        return true;
    }

    return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

3. Don’t call methods unnecessarily

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   if (firstChar >= '0' && firstChar <= '9') {
       return true;
   }

   return false;
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   return firstChar >= '0' && firstChar <= '9';
}
```

4. Reuse expressions when possible

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   return firstChar >= '0' && firstChar <= '9';
}
```

5. Use the Java API

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   return firstChar >= '0' && firstChar <= '9';
}
```

5. Use the Java API

```
public boolean imSlow(String text) {
   if (text == null) {
       return false;
   }

   if (text.length() == 0) {
       return false;
   }

   char firstChar = text.charAt(0);

   if (firstChar == '\n') {
       return true;
   }

   return Character.isDigit(firstChar);
}
```

Recap: Don’t do unnecessary work

- Don’t use exceptions unnecessarily
- Don’t check conditions unnecessarily
- Don’t call methods unnecessarily
- Reuse expressions when possible
- Use the Java API

What's inefficient about this code?

```
public void myMethod(int input) {

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

What's inefficient about this code?

```
public void myMethod(int input) {

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

What's inefficient about this code?

```
public void myMethod(int input) {

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

What's inefficient about this code?

```
public void myMethod(int input) {

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

Lazy evaluation

```
public void myMethod(int input) { // original code

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

```
public void myMethod(int input) { // improved code

    if (input < 0)
        return;
```

Lazy evaluation

```
public void myMethod(int input) { // original code

    int result = performBigCalculation();

    if (input < 0)
        return;

    . . .
```

```
public void myMethod(int input) { // improved code

    if (input < 0)
        return;

    int result = performBigCalculation();

    . . .
```

What's inefficient about this code?

```
public class DataProcessor {

    protected DataSource dataSource = new DataSource();

    public String myMethod(String arg) {
        . . . // do some work
        Result r = dataSource.query(arg);
        . . . // do some more work using r
    }

    public String anotherMethod() {
        . . . // this method does not use dataSource
    }
}
```

What's inefficient about this code?

```
public class DataProcessor {

    protected DataSource dataSource = new DataSource();

    public String myMethod(String arg) {
        . . . // do some work
        Result r = dataSource.query(arg);
        . . . // do some more work using r
    }

    public String anotherMethod() {
        . . . // this method does not use dataSource
    }
}
```

What's inefficient about this code?

```
public class DataProcessor {

    protected DataSource dataSource = new DataSource();

    public String myMethod(String arg) {
        . . . // do some work
        Result r = dataSource.query(arg);
        . . . // do some more work using r
    }

    public String anotherMethod() {
        . . . // this method does not use dataSource
    }
}
```

Lazy Initialization

```
public class DataProcessor {

    protected DataSource dataSource; // uninitialized

    public String myMethod(String arg) {
        . . . // do some work
        if (dataSource == null)
            dataSource = new DataSource();
        Result r = dataSource.query(arg);
        . . . // do some more work using r
    }

    public String anotherMethod() {
        . . . // this method does not use dataSource
    }
}
```

What's inefficient about this code?

```
public int myMethod(int x) {

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

What's inefficient about this code?

```
public int myMethod(int x) {

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

Short-circuit Boolean Operators

```
public int myMethod(int x) { // original

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

```
public int myMethod(int x) { // improved

    if (fast() || slow())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

Short-circuit Boolean Operators

```
public int myMethod(int x) { // original

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

```
public int myMethod(int x) { // improved

    if (fast() || slow()) // assuming equally likely
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

Short-circuit Boolean Operators

```
public int myMethod(int x) { // original

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

```
public int myMethod(int x) { // improved

    if (fast() || slow())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

Short-circuit Boolean Operators

```
public int myMethod(int x) { // original

    if (slow() || fast())
        x++;

    if (unlikely() || likely())
        x++;

    return x;
}
```

```
public int myMethod(int x) { // improved

    if (fast() || slow())
        x++;

    if (likely() || unlikely()) // assuming equally fast
        x++;

    return x;
}
```

Recap: Don’t do unnecessary work

- Don’t use exceptions unnecessarily
- Don’t check conditions unnecessarily
- Don’t call methods unnecessarily
- Reuse expressions when possible

- Don’t evaluate expressions until you need to
- Don’t create objects until you know you need them
- Take advantage of short-circuit operators

Code Optimization is a trade-off

- Speed/execution time
- Memory usage
- Code size
- Development effort
- Compilation time
- Understandability
- Correctness/Functionality
- Reliability
- Security

Code Optimization is a trade-off

- Speed/execution time
- Memory usage
- Code size
- Development effort
- Compilation time
- Understandability
- Correctness/Functionality
- Reliability
- Security

Don’t redo work you’ve already done

```
public class MyClass {

    public Result doCalculation(Input input) { . . . }

}
```

Use a Map to store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) { . . . }

}
```

“Wrap” the original method

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        return calculateResult(input);
    }

    private Result calculateResult(Input input) {...}

}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization: Store previous results

```
public class MyClass {

    private Map<Input, Result> results =
        new HashMap<Input, Result>();

    public Result doCalculation(Input input) {
        if (results.containsKey(input)) {
            return results.get(input);
        }
        else {
            Result result = calculateResult(input);
            results.put(input, result);
            return result;
        }
    }
    private Result calculateResult(Input input) {...}
}
```

Memoization

- Can greatly improve execution time assuming:
  - Result is calculated for same Input more than once
  - Execution time of calculateResult is significantly greater than that of Map lookup and storing Result

- Other tradeoffs:
  - Memory usage can increase greatly
  - May need to control the number of Results in Map

SD2x4.9

Threads

Chris

Thread-level Parallelism

- Modern programming languages support **threads**, which allow a program to do more than one thing at a time

- Examples
  - Reading data from one source while writing to another
  - Performing two parts of a calculation simultaneously
  - Blocking/waiting for some input while performing some calculation

Threads

- Threads: separate lines of execution in the same process (running program)

- Each thread has its own:
  - stack (local variables, method arguments)
  - program counter (address of next instruction to be executed)

- Threads in the same process share:
  - heap (space in memory where objects are stored)
  - static/global variables

Threads in Java

- **java.lang.Thread**

- Represents the starting point of a separate line of execution in the running program

- **start:** launch a new line of execution

- **run:** entry point to new line of execution

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread t = new MyThread();
```

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread t = new MyThread();

t.start(); // launches new thread and invokes t.run()
```

```
public class MyThread extends Thread {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread t = new MyThread();

t.start(); // launches new thread and invokes t.run()

// this code continues while t.run() executes simultaneously
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread mt = new MyThread();
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread mt = new MyThread();

Thread t = new Thread(mt);
```

```
public class MyThread extends SomethingElse implements Runnable {

   public MyThread( ) { . . . . }

   public void run( ) { . . . . } // entry point of new thread

   . . . .
}
```

```
MyThread mt = new MyThread();

Thread t = new Thread(mt);

t.start(); // launches new thread and invokes mt.run()
```

```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

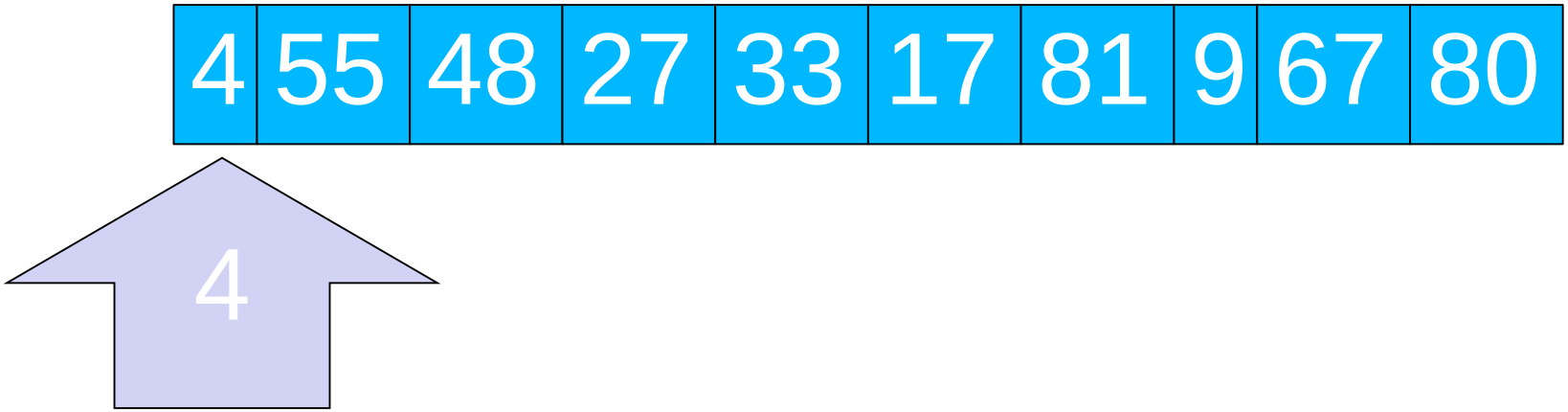
```
MyThread t = new MyThread(10);
```

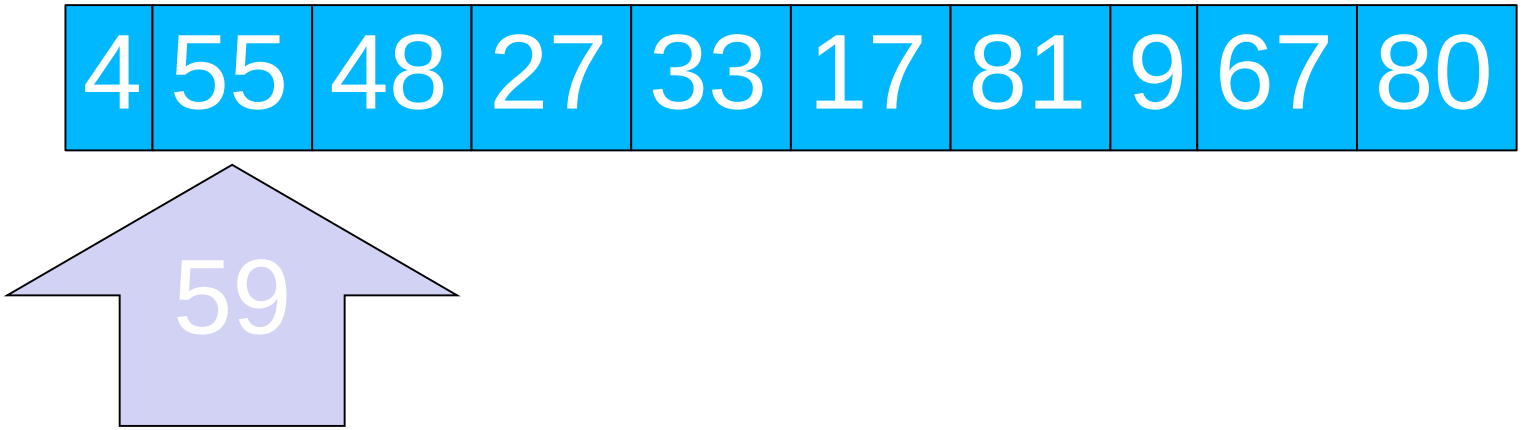
```
public class MyThread extends Thread {

   private int howMany;

   public MyThread(int howMany) {
       this.howMany = howMany;
   }

   public void run( ) {
       for (int i = 0; i < howMany; i++) {
           System.out.println(i);
       }
   }

   . . . .
}
```

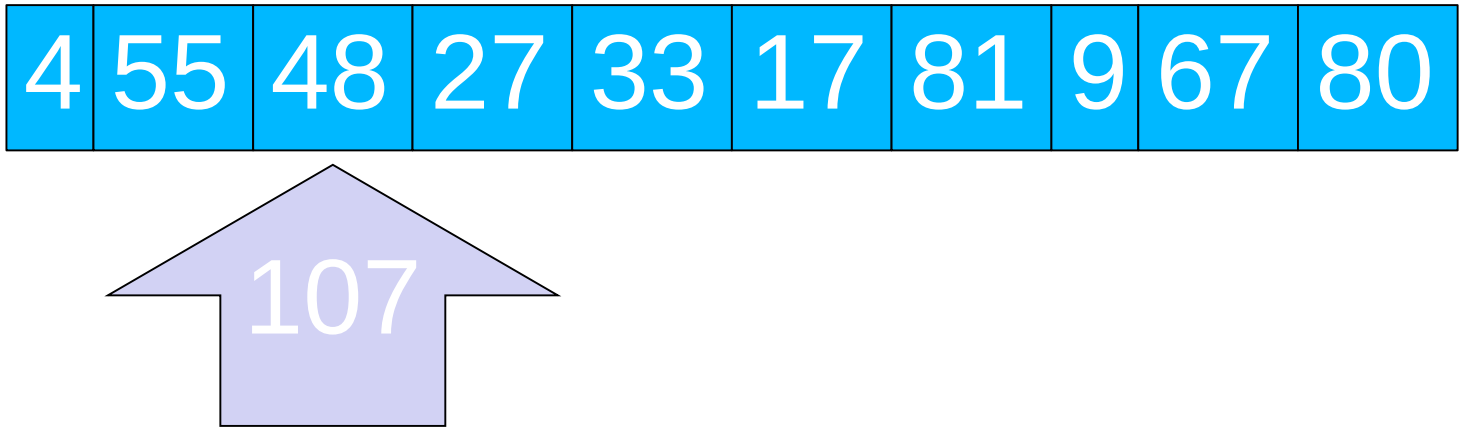
```
MyThread t = new MyThread(10);

t.start();
```

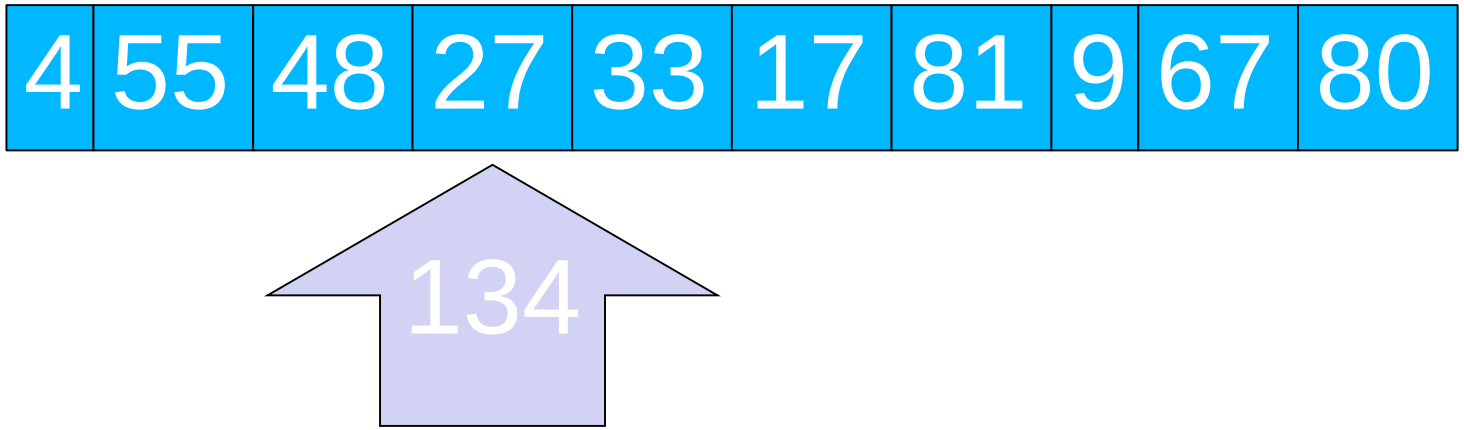
How can we use threads to address “embarrassingly parallel” problems?

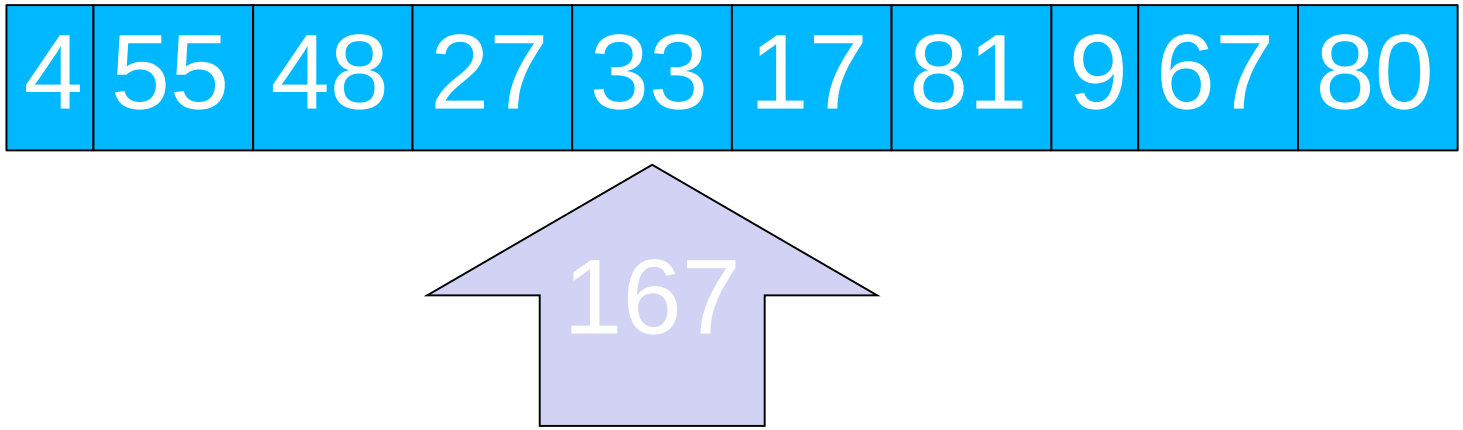
4	55	48	27	33	17	81	9	67	80

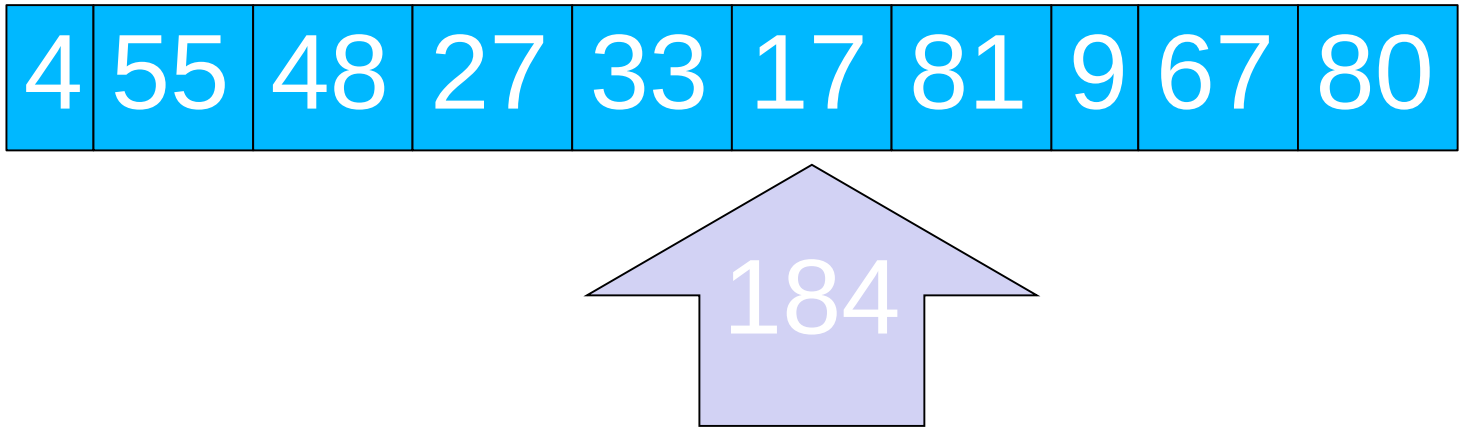

4 55 48 27 33 17 81 9 67 80
4

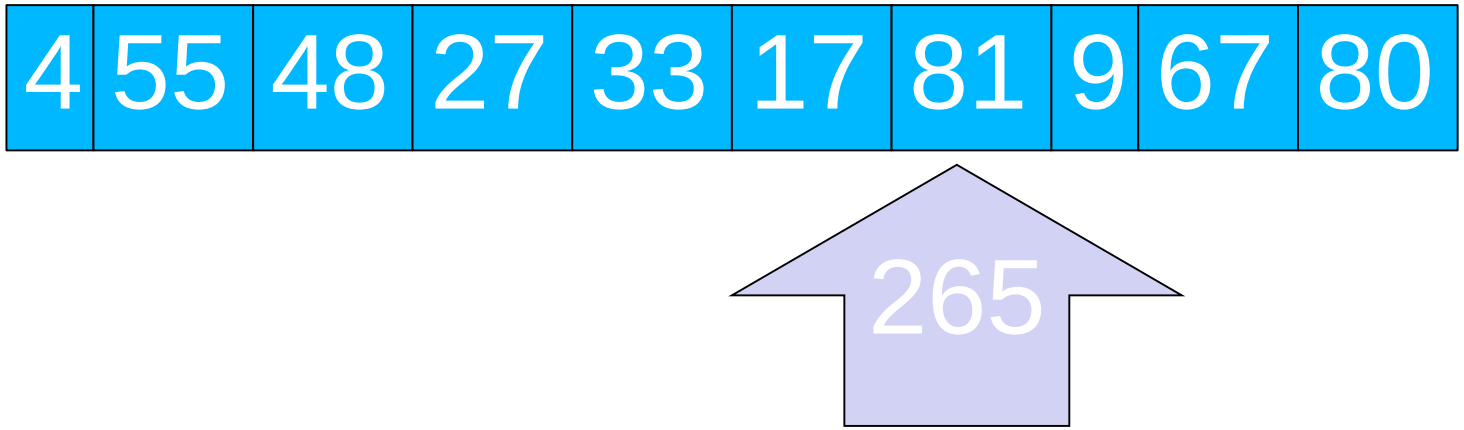

4
55
48
27
33
17
81
9
67
80
59

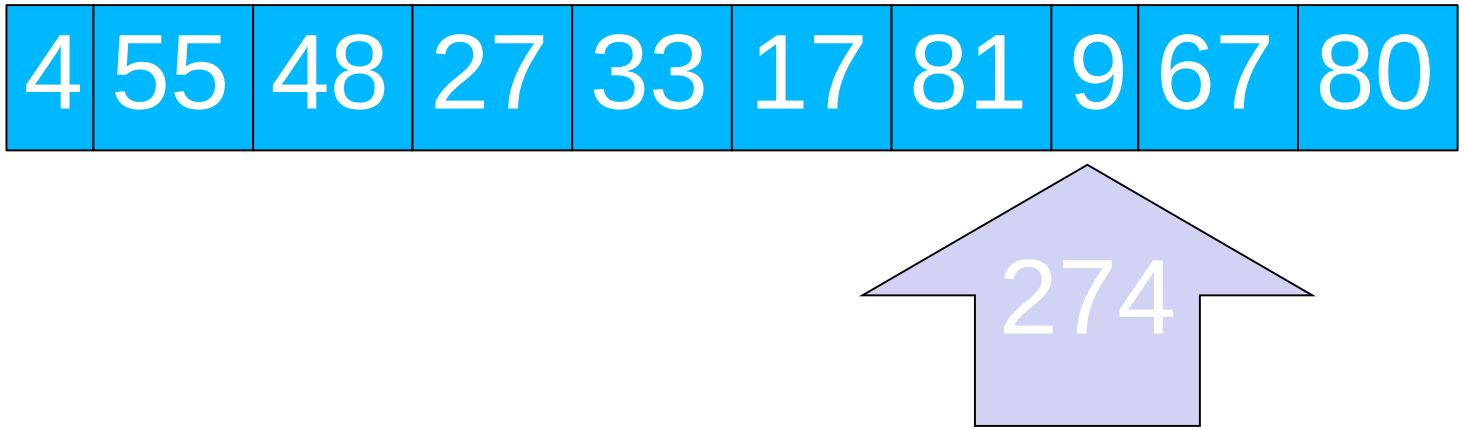

4
55
48
27
33
17
81
9
67
80
107

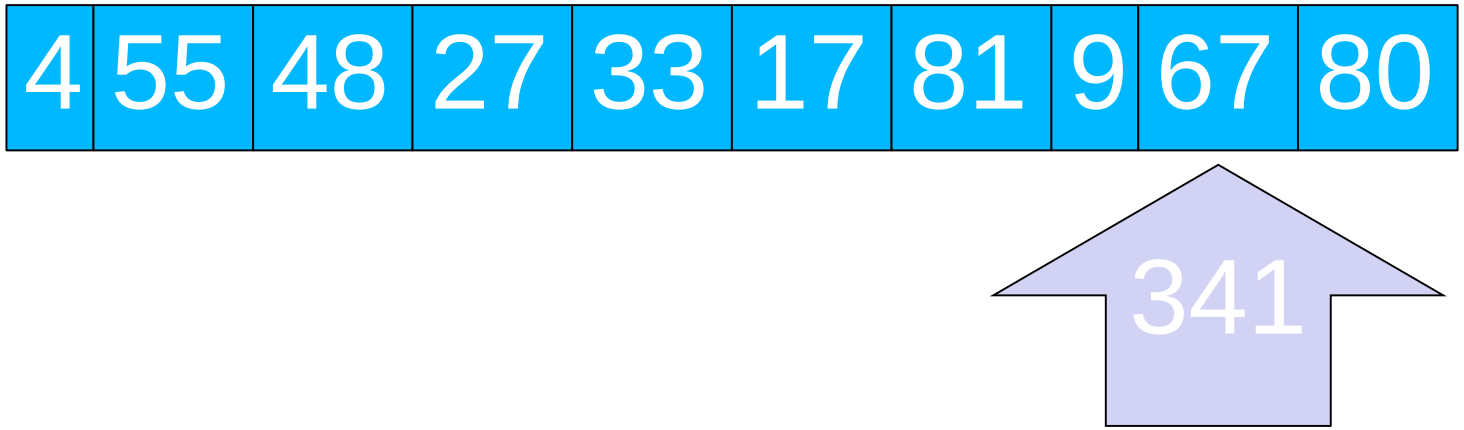

4
55
48
27
33
17
81
9
67
80
134

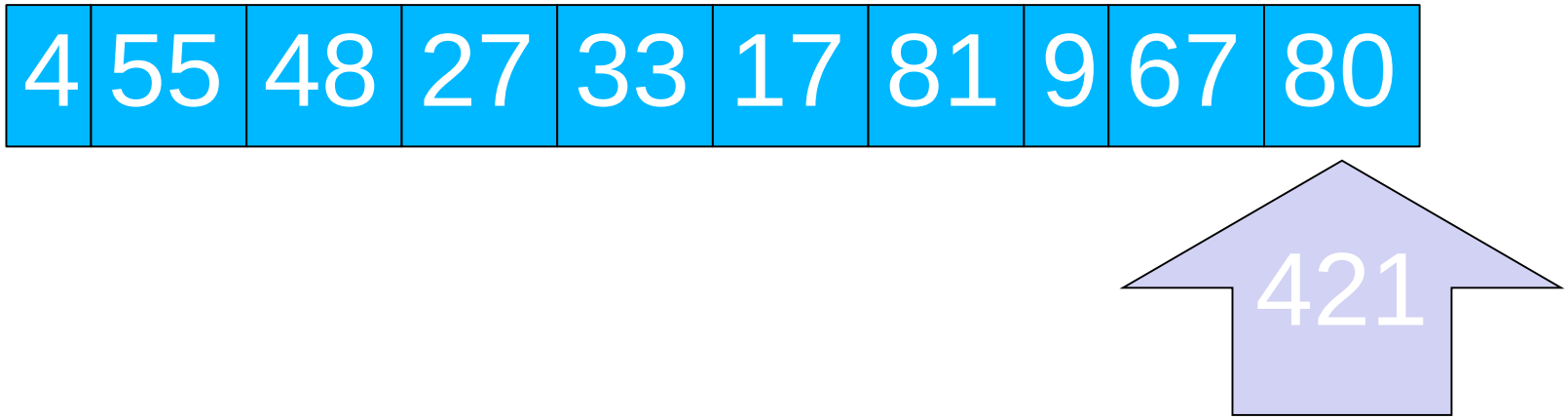

4
55
48
27
33
17
81
9
67
80
167


4
55
48
27
33
17
81
9
67
80
184


4
55
48
27
33
17
81
9
67
80
265


4
55
48
27
33
17
81
9
67
80
274


4
55
48
27
33
17
81
9
67
80
341


4
55
48
27
33
17
81
9
67
80
421

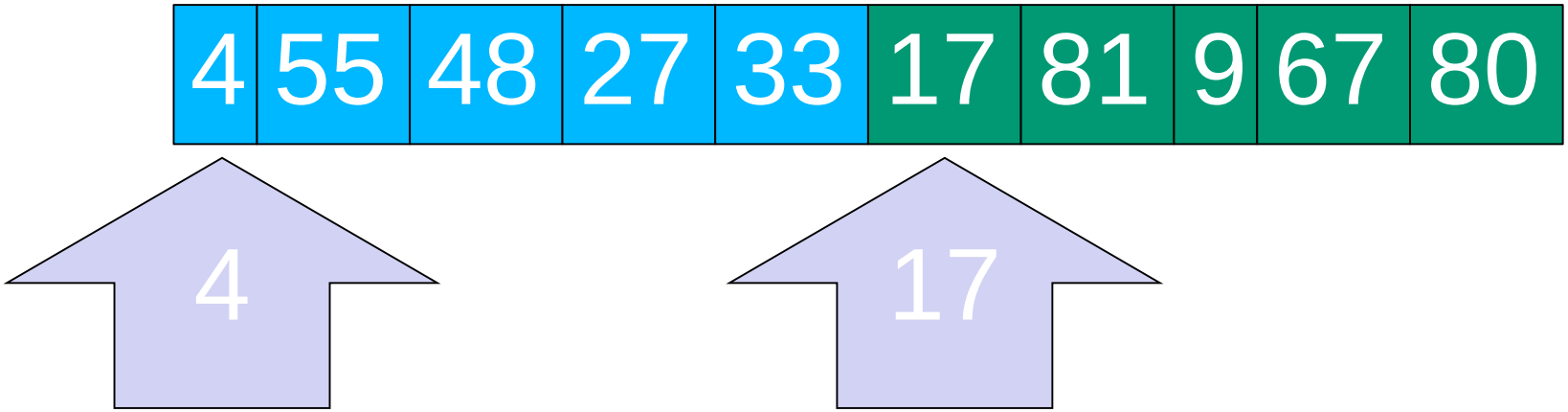
4	55	48	27	33	17	81	9	67	80

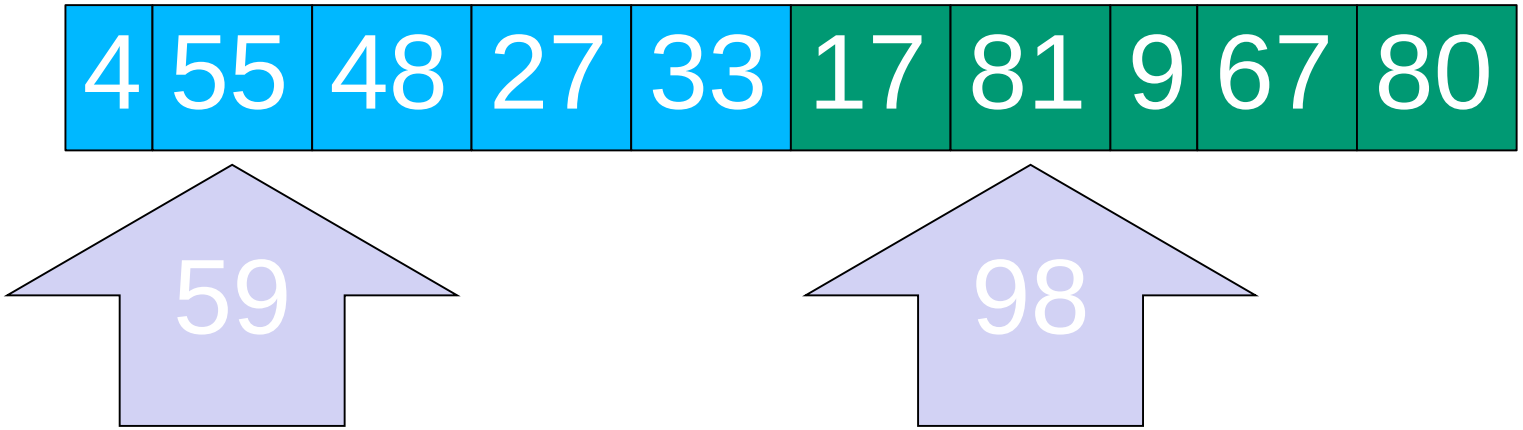
421

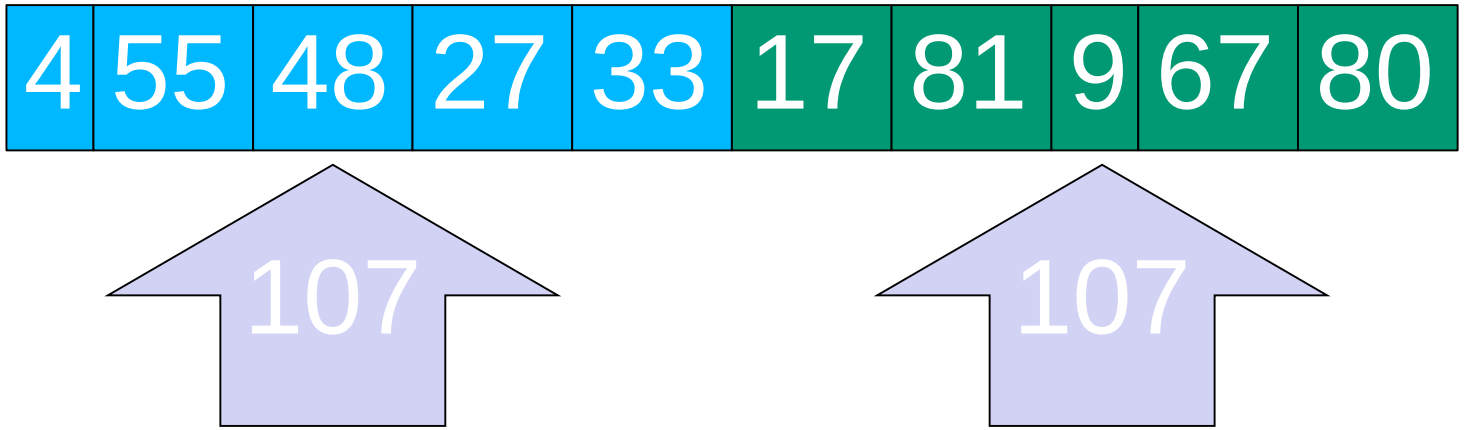
421

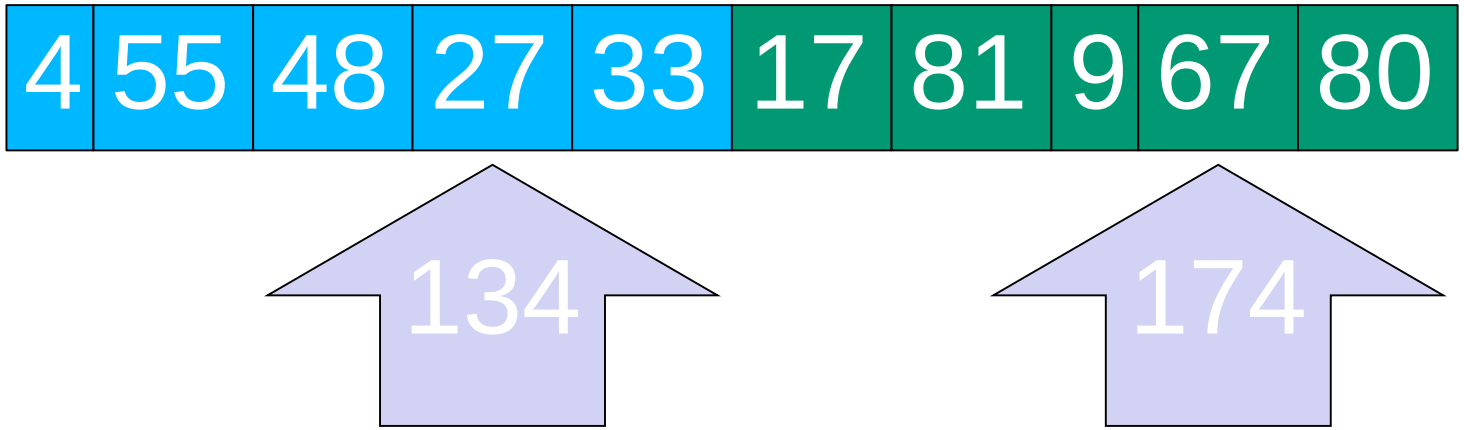
4	55	48	27	33	17	81	9	67	80

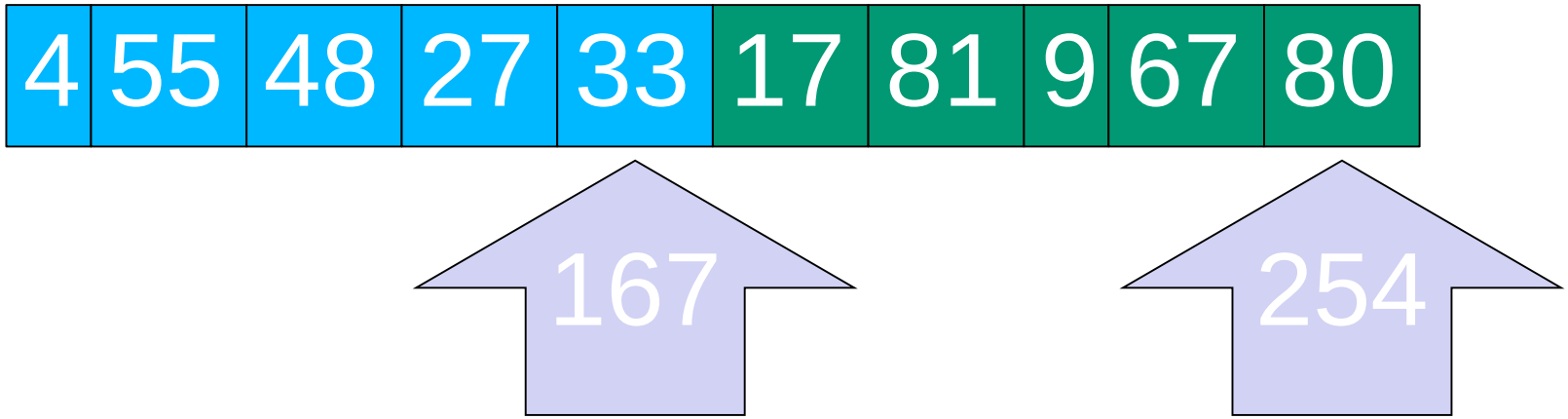
4	55	48	27	33	17	81	9	67	80

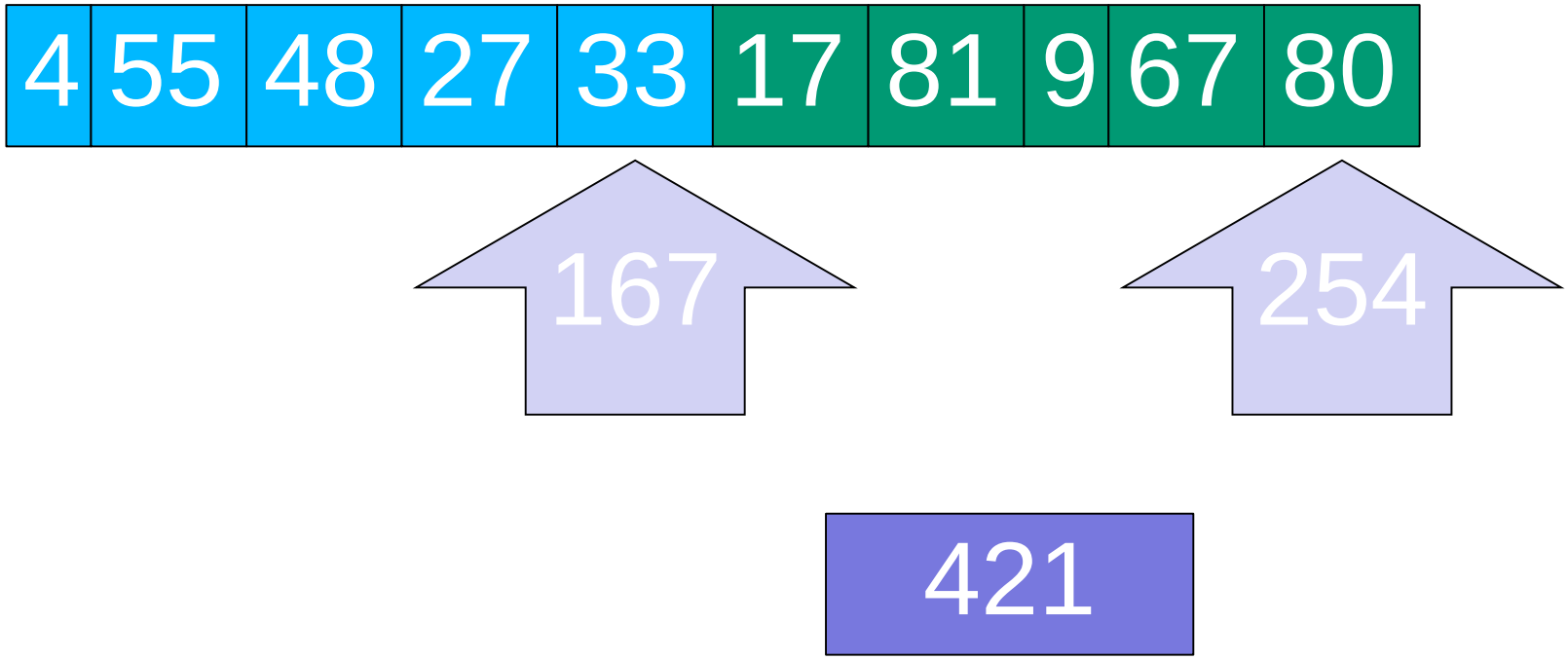

4
55
48
27
33
17
81
9
67
80
4
17


4
55
48
27
33
17
81
9
67
80
59
98


4
55
48
27
33
17
81
9
67
80
107
107


4
55
48
27
33
17
81
9
67
80
134
174


4
55
48
27
33
17
81
9
67
80
167
254


4
55
48
27
33
17
81
9
67
80
167
254
421

```
public class AdderThread extends Thread {

   private int[] array;
   private int sum;

   public MyThread(int[] array) {
       this.array = array;
   }

   public void run( ) {
       for (int i = 0; i < array.length; i++) {
           sum += array[i];
       }
   }

   public int getSum() {
       return sum;
   }
}
```

```java
public class AdderThread extends Thread {

   private int[] array;
   private int sum;

   public MyThread(int[] array) {
       this.array = array;
   }

   public void run( ) {
       for (int i = 0; i < array.length; i++) {
           sum += array[i];
       }
   }

   public int getSum() {
       return sum;
   }
}
```

```java
public class AdderThread extends Thread {

   private int[] array;
   private int sum;

   public MyThread(int[] array) {
       this.array = array;
   }

   public void run( ) {
       for (int i = 0; i < array.length; i++) {
           sum += array[i];
       }
   }

   public int getSum() {
       return sum;
   }
}
```

```java
public class AdderThread extends Thread {

   private int[] array;
   private int sum;

   public MyThread(int[] array) {
       this.array = array;
   }

   public void run( ) {
       for (int i = 0; i < array.length; i++) {
           sum += array[i];
       }
   }

   public int getSum() {
       return sum;
   }
}
```

```java
public class AdderThread extends Thread {

   private int[] array;
   private int sum;

   public MyThread(int[] array) {
       this.array = array;
   }

   public void run( ) {
       for (int i = 0; i < array.length; i++) {
           sum += array[i];
       }
   }

   public int getSum() {
       return sum;
   }
}
```

```
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                                array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```java
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                          array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                                array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```java
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                          array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                                array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                                array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();




    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```java
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                          array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();

    firstAdder.join(); // wait here for the threads to finish!
    secondAdder.join();

    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```java
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                          array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();

    firstAdder.join(); // wait here for the threads to finish!
    secondAdder.join();

    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

```java
public int addArray(int[] array) {

    int midpoint = array.length / 2;

    // split array in half
    int[] firstHalf = Arrays.copyOfRange(array, 0, midpoint);
    int[] secondHalf = Arrays.copyOfRange(array, midpoint,
                                          array.length);

    AdderThread firstAdder = new AdderThread(firstHalf);
    AdderThread secondAdder = new AdderThread(secondHalf);

    firstAdder.start();
    secondAdder.start();

    firstAdder.join(); // wait here for the threads to finish!
    secondAdder.join();

    int firstSum = firstAdder.getSum();
    int secondSum = secondAdder.getSum();

    return firstSum + secondSum;

}
```

Recap: Threads

- Threads: separate lines of execution in the same process (running program)

- Extend the **Thread** class (or implement **Runnable**) and implement the **run** method as the entry point for a new thread

- We can use threads to improve the efficiency of our code in addressing “embarrassingly parallel” problems

SD2x4.10

Race Conditions

Chris

What can go wrong when using threads?

```
public class BankAccount {
    private double balance = 0;

    public void setBalance(double amount) {
         balance = amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void setBalance(double amount) {
         balance = amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void setBalance(double amount) {
        balance = amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void setBalance(double amount) {
         balance = amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
         this.account = account;
    }

    public void run() {
         account.setBalance(100);
    }
}
```

```java
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
         this.account = account;
    }

    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    private BankAccount account;

    public MyThread2(BankAccount account) {
         this.account = account;
    }

    public void run() {
         account.setBalance(200);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    private BankAccount account;

    public MyThread2(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(200);
    }
}
```

```
public class MyThread1 extends Thread {
    private BankAccount account;

    public MyThread1(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    private BankAccount account;

    public MyThread2(BankAccount account) {
        this.account = account;
    }

    public void run() {
        account.setBalance(200);
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // what gets printed?
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```java
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```java
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```java
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100); // now balance = 100
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200); // now balance is 200
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 200
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```java
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```java
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```java
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```java
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100);
    }
}
```

```java
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200); // now balance is 200
    }
}
```

```java
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.setBalance(100); // now balance is 100
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.setBalance(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.setBalance(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 100
```

What could happen that’s even worse?

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
         balance += amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
        balance += amount;
    }

    public double getBalance() {
        return balance;
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100); // now balance is 0 + 100 = 100
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // now balance is 100 + 200 = 300
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 300
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```java
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```java
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```java
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```java
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```java
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // now balance is 0 + 200 = 200
    }
}
```

```java
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100); // now balance is 200 + 100 = 300
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 300
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 100??
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // prints 100?? 200??
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance += 100;
```

```
MyThread1.run():
  myAccount.deposit(100):
    balance += 100;
```

```
MyThread2.run():
  myAccount.deposit(200):
    balance += 200;
```

```
MyThread1.run():
  myAccount.deposit(100):
    balance += 100;
```

```
MyThread2.run():
  myAccount.deposit(200):
    balance += 200;
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

```
MyThread1.run():
  myAccount.deposit(100):
    balance = balance + 100;
```

```
MyThread2.run():
  myAccount.deposit(200):
    balance = balance + 200;
```

```
MyThread1.run():
  myAccount.deposit(100):
    balance = balance + 100;
```

1. Read

```
MyThread2.run():
  myAccount.deposit(200):
    balance = balance + 200;
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

1. Read

balance 0

```
MyThread1.run():
  myAccount.deposit(100):
    balance = balance + 100;
```

```
MyThread2.run():
  myAccount.deposit(200):
    balance = balance + 200;
```

1. Read

local: 0

1. Read

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

local: 100

2. Add

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

local: 100

2. Add

3. Write

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

2. Add

3. Write

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

3. Write

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

1. Read

2. Add

3. Write

1. Read

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

1. Read

2. Add

3. Write

1. Read

local: 100

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

1. Read

2. Add

3. Write

1. Read

2. Add

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

1. Read

2. Add

3. Write

1. Read

2. Add

local: 300

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

2. Add

3. Write

1. Read

2. Add

3. Write

local: 300

balance 100

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

2. Add

3. Write

1. Read

2. Add

3. Write

balance 300

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

2. Add

3. Write

1. Read

2. Add

3. Write

balance 300

```
MyThread1.run():
  myAccount.deposit(100):
    balance = balance + 100;
```

```
MyThread2.run():
  myAccount.deposit(200):
    balance = balance + 200;
```

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

local: 0

1. Read

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

local: 100

2. Add

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

1. Read

1. Read

local: 100

2. Add

1. Read

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

1. Read

1. Read

local: 100

2. Add

1. Read

local: 0

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

1. Read

2. Add

1. Read

local: 100

2. Add

1. Read

2. Add

local: 200

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

local: 100

2. Add

1. Read

2. Add

local: 200

3. Write

balance 0

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read
2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write
1. Read
2. Add

1. Read
2. Add

local: 100

1. Read
2. Add
3. Write

balance 200

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

1. Read
2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write
1. Read
2. Add

1. Read

local: 100

2. Add

1. Read

2. Add

3. Write

balance 200

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

local: 100

2. Add

1. Read

2. Add

3. Write

3. Write

balance 200

```
MyThread1.run():

  myAccount.deposit(100):

    balance = balance + 100;
```

3. Write

1. Read

2. Add

```
MyThread2.run():

  myAccount.deposit(200):

    balance = balance + 200;
```

3. Write

1. Read

2. Add

1. Read

2. Add

1. Read

2. Add

3. Write

3. Write

balance 100

Recap: Race Conditions

- We cannot always control or even know the order in which thread operations will be executed

- This may lead to non-deterministic results in some cases
- And race conditions (and erroneous results) in others

SD2x4.11

Thread Synchronization

Chris

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
         balance += amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
        balance += amount; // race condition can exist here
    }

    public double getBalance() {
        return balance;
    }
}
```

Addressing Race Conditions

- The problem is that two threads were simultaneously trying to read then update a variable
- This wouldn't be a problem if we made the read/update combination **"atomic"**
- We can do this by only allowing **one thread at a time** to execute that instruction

Synchronized methods

- If a method is **synchronized**, only one thread at a time may execute that method on that object
- (Or any other synchronized methods on that object)

- If a thread is executing a synchronized method on an object, any other thread that tries to execute a synchronized method on the same object will need to wait for the first thread to finish

- This is known as **mutual exclusion**

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
         balance += amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         balance += amount;
    }

    public double getBalance() {
         return balance;
    }
}
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // cannot start until t1 finishes!
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100); // now balance is 0 + 100 = 100
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // cannot start until t1 finishes!
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // only runs after t1 finishes
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
        account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
        account.deposit(200); // now balance is 100 + 200 = 300
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance());
```

```
public class MyThread1 extends Thread {
    . . .
    public void run() {
         account.deposit(100);
    }
}
```

```
public class MyThread2 extends Thread {
    . . .
    public void run() {
         account.deposit(200);
    }
}
```

```
BankAccount myAccount = new BankAccount();
MyThread1 t1 = new MyThread1(myAccount);
MyThread2 t2 = new MyThread2(myAccount);

t1.start();
t2.start();
t1.join();
t2.join();

System.out.println(myAccount.getBalance()); // always prints 300
```

Why synchronized methods are perhaps a bit too much

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         if (amount < 0)
             return;
         Log.log("depositing " + amount);
         balance += amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         if (amount < 0)
             return;
         Log.log("depositing " + amount);
         balance += amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         if (amount < 0)
             return;
         Log.log("depositing " + amount);
         balance += amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
        if (amount < 0)
            return;
        Log.log("depositing " + amount);
        balance += amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
         if (amount < 0)
             return;
         Log.log("depositing " + amount);
         synchronized (this) {
             balance += amount;
         }
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
        if (amount < 0)
            return;
        Log.log("depositing " + amount);
        synchronized (this) {
            balance += amount;
        }
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public void deposit(double amount) {
        if (amount < 0)
            return;
        Log.log("depositing " + amount);
        synchronized (this) {
            balance += amount;
        }
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         balance += amount;
    }

    public synchronized void withdraw(double amount) {
         balance -= amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
         balance += amount;
    }

    public synchronized void withdraw(double amount) {
         balance -= amount;
    }
}
```

```
public class BankAccount {
    private double balance = 0;

    public synchronized void deposit(double amount) {
        balance += amount;
    }

    public synchronized void withdraw(double amount) {
        balance -= amount;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;

    public synchronized void incrementCount1() {
         count1++;
    }

    public synchronized void incrementCount2() {
         count2++;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;

    public synchronized void incrementCount1() {
        count1++;
    }

    public synchronized void incrementCount2() {
        count2++;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;

    public synchronized void incrementCount1() {
         count1++;
    }

    public synchronized void incrementCount2() {
         count2++;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;

    public synchronized void incrementCount1() {
         count1++;
    }

    public synchronized void incrementCount2() {
         count2++;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;

    public synchronized void incrementCount1() {
         count1++;
    }

    public synchronized void incrementCount2() {
         count2++;
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;
    private Object lock1 = new Object(), lock2 = new Object();

    public void incrementCount1() {
         synchronized (lock1) {
             count1++;
         }
    }

    public void incrementCount2() {
         synchronized (lock2) {
             count2++;
         }
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;
    private Object lock1 = new Object(), lock2 = new Object();

    public void incrementCount1() {
         synchronized (lock1) {
             count1++;
         }
    }

    public void incrementCount2() {
         synchronized (lock2) {
             count2++;
         }
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;
    private Object lock1 = new Object(), lock2 = new Object();

    public void incrementCount1() {
        synchronized (lock1) {
            count1++;
        }
    }

    public void incrementCount2() {
        synchronized (lock2) {
            count2++;
        }
    }
}
```

```
public class Counter {
    private int count1 = 0;
    private int count2 = 0;
    private Object lock1 = new Object(), lock2 = new Object();

    public void incrementCount1() {
        synchronized (lock1) {
            count1++;
        }
    }

    public void incrementCount2() {
        synchronized (lock2) {
            count2++;
        }
    }
}
```

Recap: Synchronization

- Synchronization, or "**mutual exclusion**," can be used to address race conditions
- **Synchronized methods** ensure that only one thread at a time can enter those methods
- Use **synchronized blocks** for more fine-grained access

Recap: Software Maintenance

- **Internal quality:** improve the design
  - Easier to read and understand
  - Easier to change
  - Easier to test

- **External quality**
  - Find and fix bugs
  - More resistant to problems
  - More efficient in terms of time and space