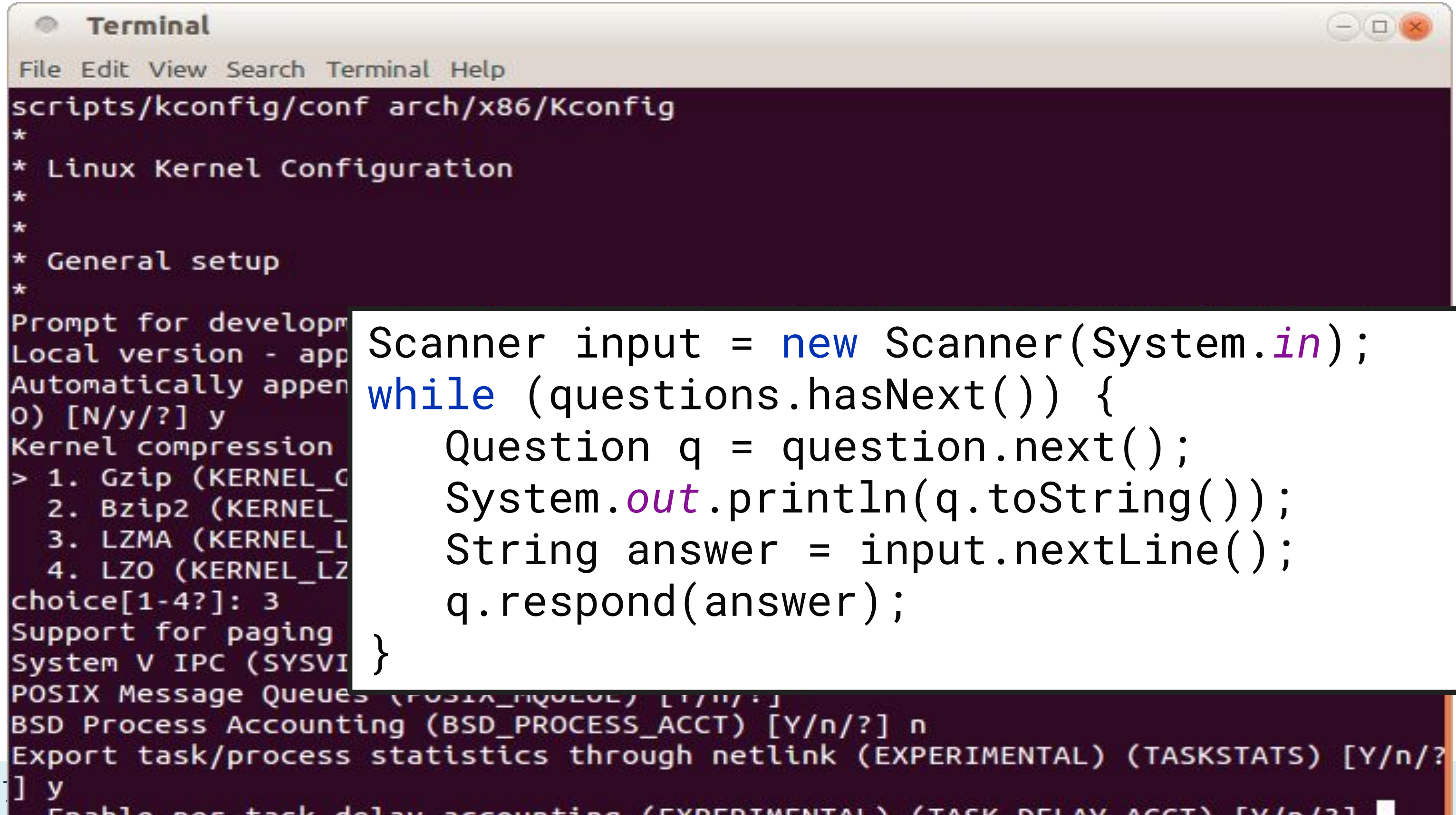
Principles of Software Construction: Objects, Design, and Concurrency

Asynchrony and Concurrency

Claire Le Goues **Bogdan Vasilescu**

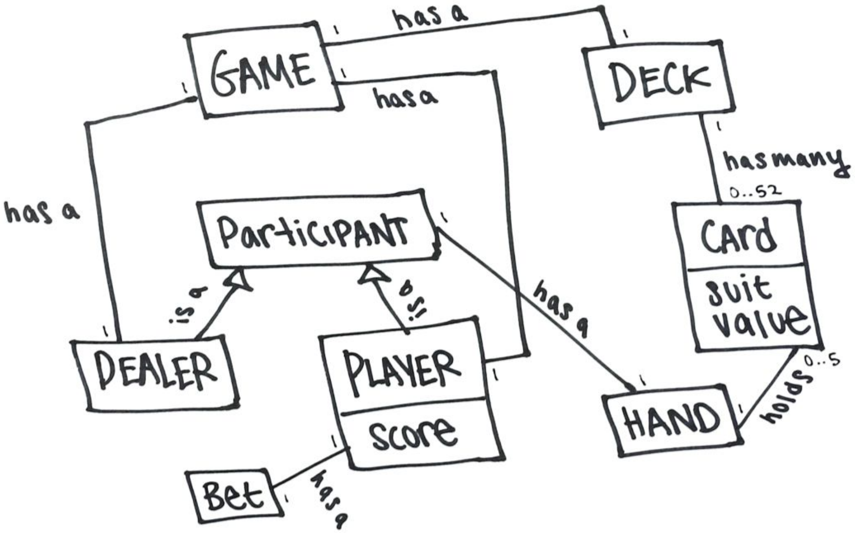
Interaction with CLI



```
scripts/kconfig/conf arch/x86/Kconfig
*
* Linux Kernel Configuration
*
*
* General setup
*
Prompt for developm
Local version - app
Automatically appen
O) [N/y/?] y
Kernel compression
> 1. Gzip (KERNEL_G
  2. Bzip2 (KERNEL_
  3. LZMA (KERNEL_L
  4. LZO (KERNEL_LZ
choice[1-4?]: 3
Support for paging
System V IPC (SYSVI
POSIX Message Queues (POSIX_MQUEUE) [Y/n/?]
BSD Process Accounting (BSD_PROCESS_ACCT) [Y/n/?] n
Export task/process statistics through netlink (EXPERIMENTAL) (TASKSTATS) [Y/n/?
] y
Enable per-task delay accounting (EXPERIMENTAL) (TASK_DELAY_ACCT) [Y/n/?]
```

```java
Scanner input = new Scanner(System.in);
while (questions.hasNext()) {
   Question q = question.next();
   System.out.println(q.toString());
   String answer = input.nextLine();
   q.respond(answer);
}
```

A backend with no interaction

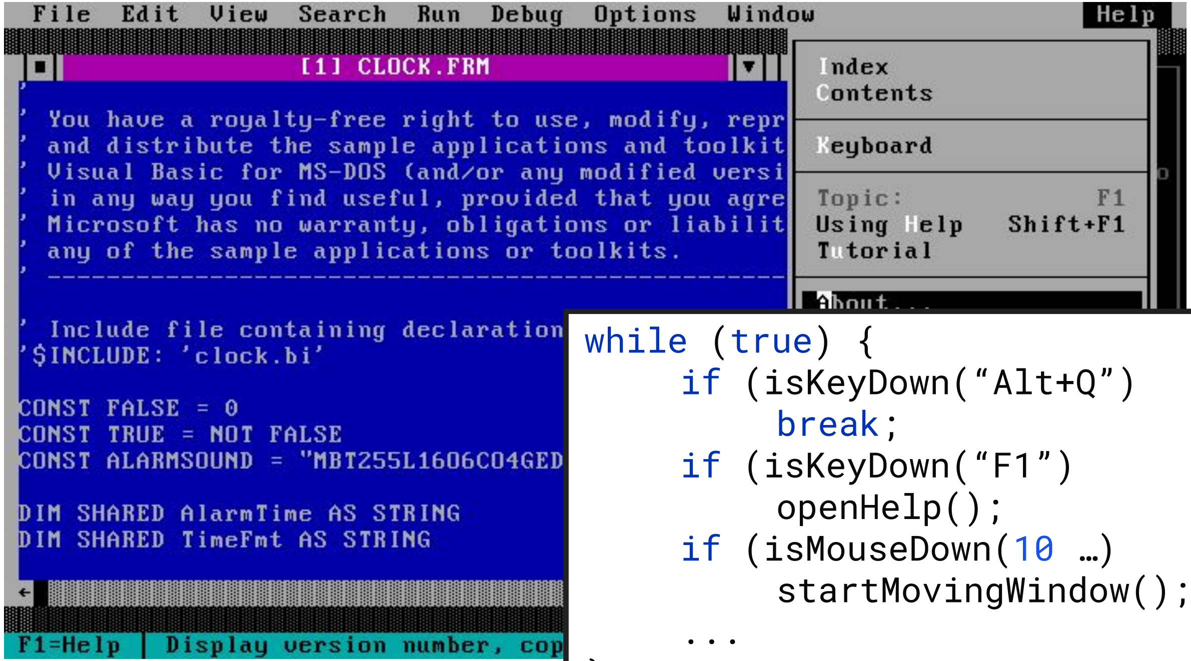


One Possible Domain model

this is Not a reference solution, it's an example of what a domain model looks like

What have we not yet seen?

How do you wait?

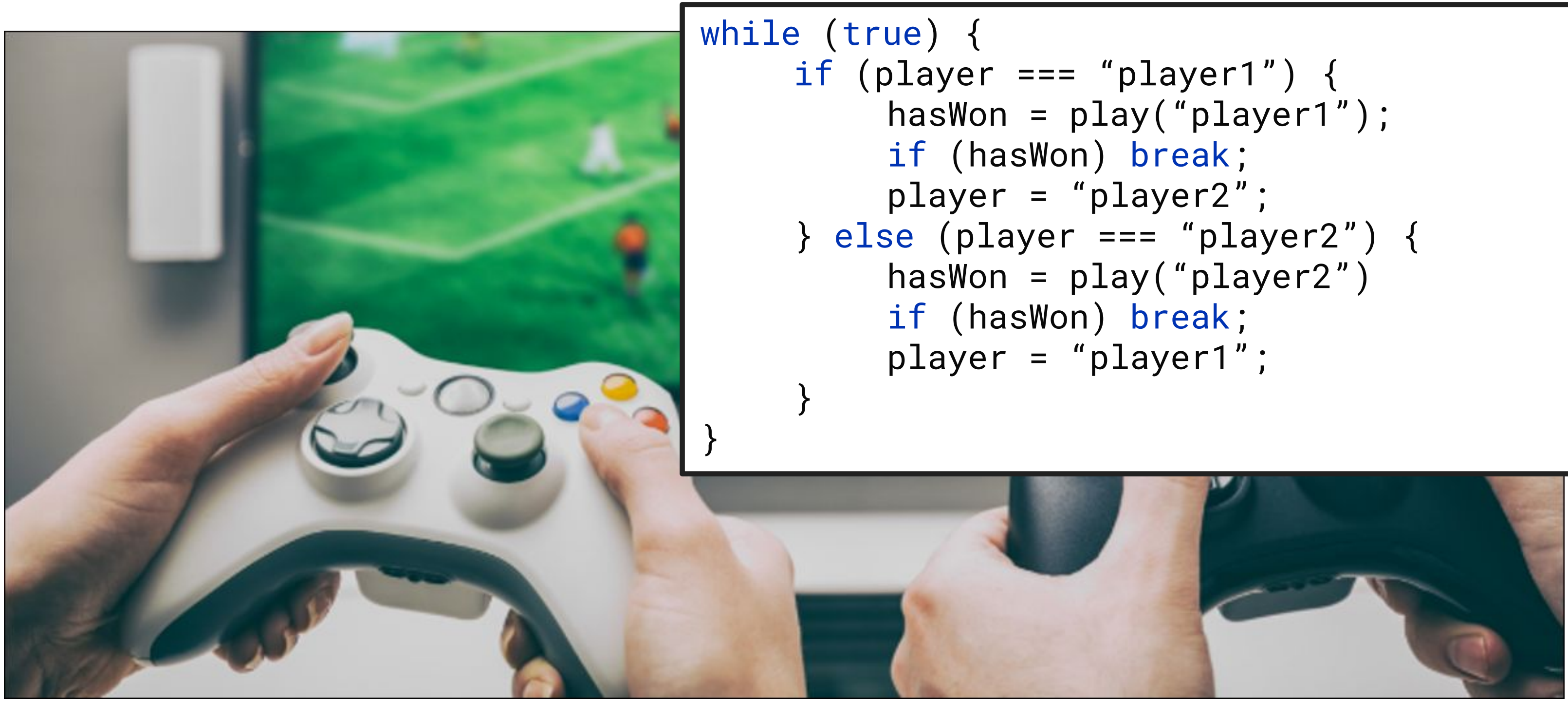


```
'
' You have a royalty-free right to use, modify, repr
' and distribute the sample applications and toolkit
' Visual Basic for MS-DOS (and/or any modified versi
' in any way you find useful, provided that you agre
' Microsoft has no warranty, obligations or liabilit
' any of the sample applications or toolkits.
'
' -----------------------------------------------

' Include file containing declaration
'$INCLUDE: 'clock.bi'

CONST FALSE = 0
CONST TRUE = NOT FALSE
CONST ALARMSOUND = "MBT255L1606C04GED

DIM SHARED AlarmTime AS STRING
DIM SHARED TimeFmt AS STRING
```

F1=Help | Display version number, cop

```
while (true) {
    if (isKeyDown(“Alt+Q”)
        break;
    if (isKeyDown(“F1”)
        openHelp();
    if (isMouseDown(10 …)
        startMovingWindow();
    ...
}
```

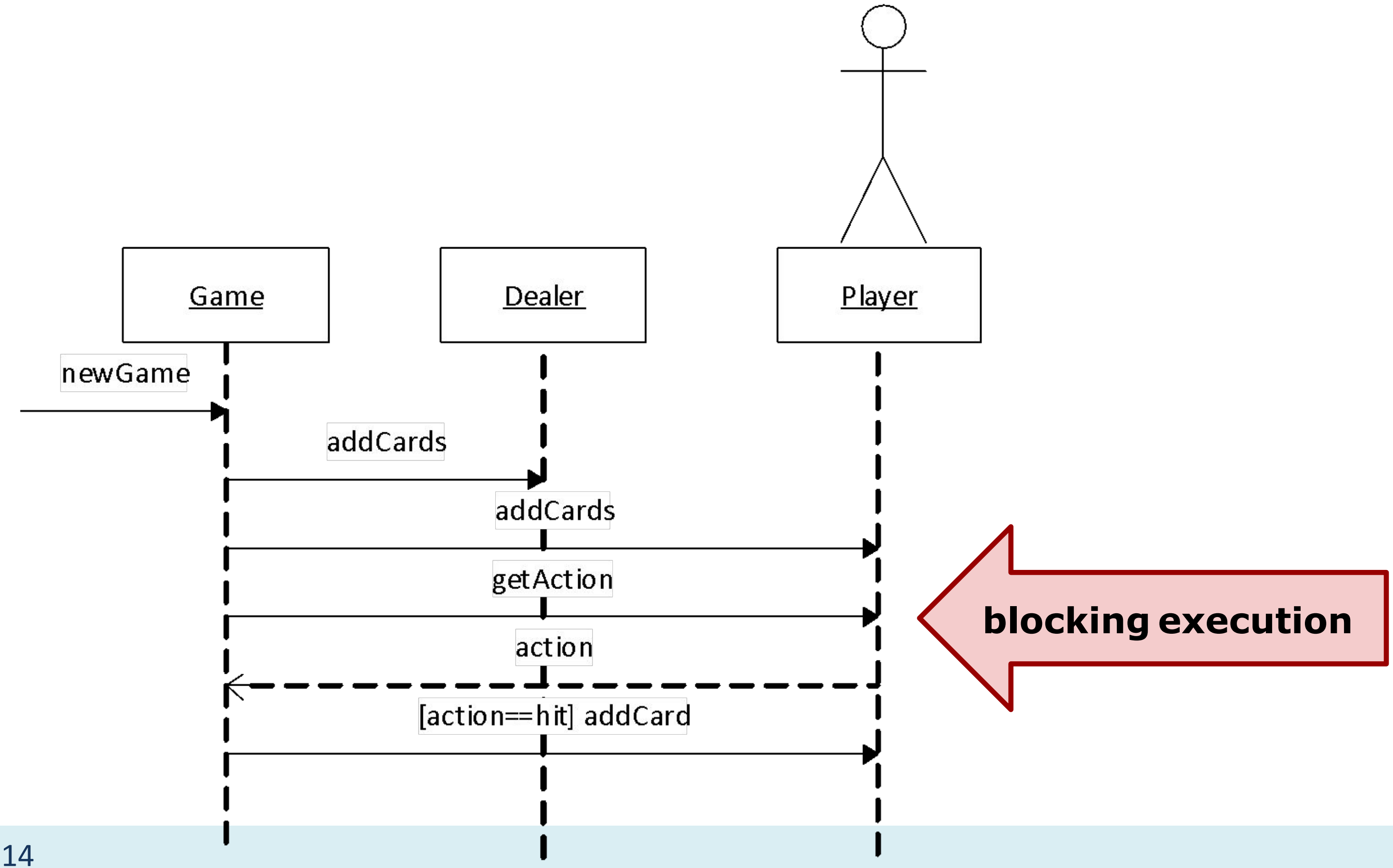
How do you multi-player?



```
while (true) {
    if (player === “player1”) {
        hasWon = play(“player1”);
        if (hasWon) break;
        player = “player2”;
    } else (player === “player2”) {
        hasWon = play(“player2”)
        if (hasWon) break;
        player = “player1”;
    }
}
```

https://www.cloudsavvyit.com/2586/how-to-build-your-multiplayer-games-server-architecture/

Potential issue: Blocking interactions with users



Today

Beyond serial execution

- Intro to Concurrency
- Event-based Programming
- I/O, GUIs
- Observer Pattern

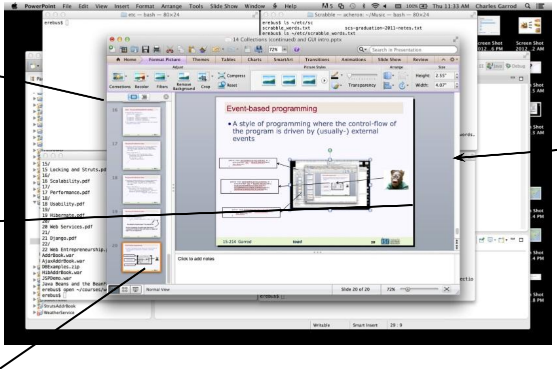
Event-based programming

- Style of programming where control-flow is driven by (usually external) events

```
public void performAction(ActionEvent e) {
    List<String> lst = Arrays.asList(bar);
    foo.peek(42)
}
```

```
public void performAction(ActionEvent e) {
    bigBloatedPowerPointFunction(e);
    withANameSoLongIMadeItTwoMethods(e);
    yesIKnowJavaDoesntWorkLikeThat(e);
}
```

```
public void performAction(ActionEvent e) {
    List<String> lst = Arrays.asList(bar);
    foo.peek(40)
}
```



PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
Thu 11:33 AM Charles Garrod
etc — bash — 80×24
Scrabble — acheron: ~/Music — bash — 80×24
14 Collections (continued) and GUI intro.pptx
Search in Presentation
Home Format Picture Themes Tables Charts SmartArt Transitions Animations Slide Show Review
Adjust Picture Styles Arrange Size
Corrections Recolor Filters Remove Background Crop Compress Reset Transparency
Height: 2.55" Width: 4.07"
Click to add notes
Normal View Slide 20 of 20 72%
Writable Smart Insert 29 : 9

```
erebus$ ls ~/etc/sc
scrabble_words.txt          scs-graduation-2011-notes.txt
erebus$ ls ~/etc/scrabble_words.txt
```

```
15/
15 Locking and Struts.pd
16/
16 Scalability.pdf
17/
17 Performance.pdf
18/
18 Usability.pdf
19/
19 Hibernate.pdf
20/
20 Web Services.pdf
21/
21 Django.pdf
22/
22 Web Entrepreneurship.
AddrBook.war
AjaxAddrBook.war
DBExamples.zip
HibAddrBook.war
JSPDemo.war
Java Beans and the BeanF
erebus$ open ~/courses/w
erebus$
```

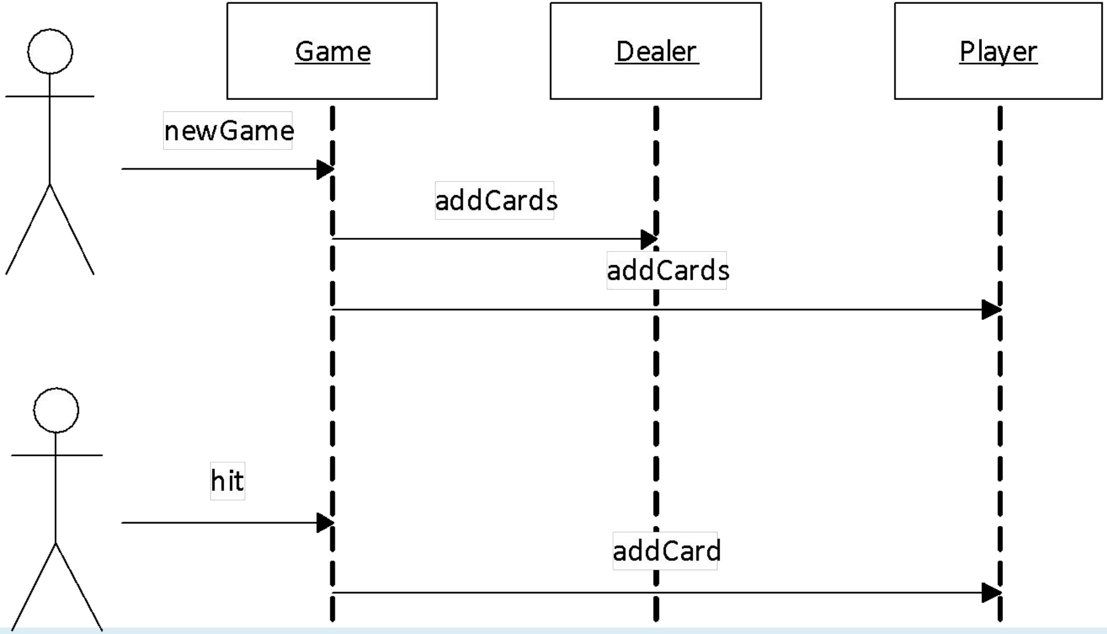
Event-based programming

- A style of programming where the control-flow of the program is driven by (usually-) external events

15-214 Garrod toad 20

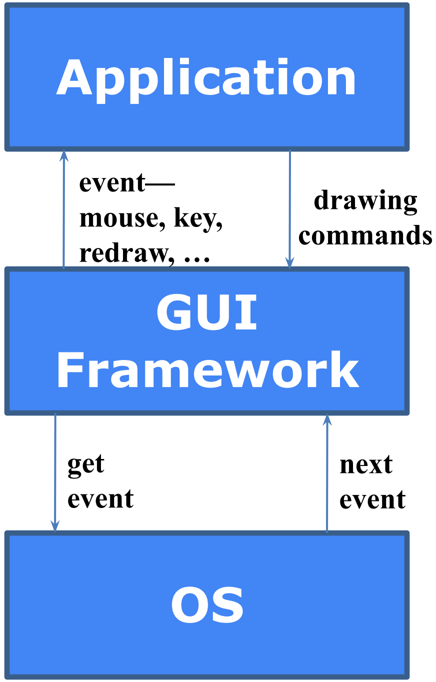
Interactions with users through events

- Do not block waiting for user response
- Instead, react to user events

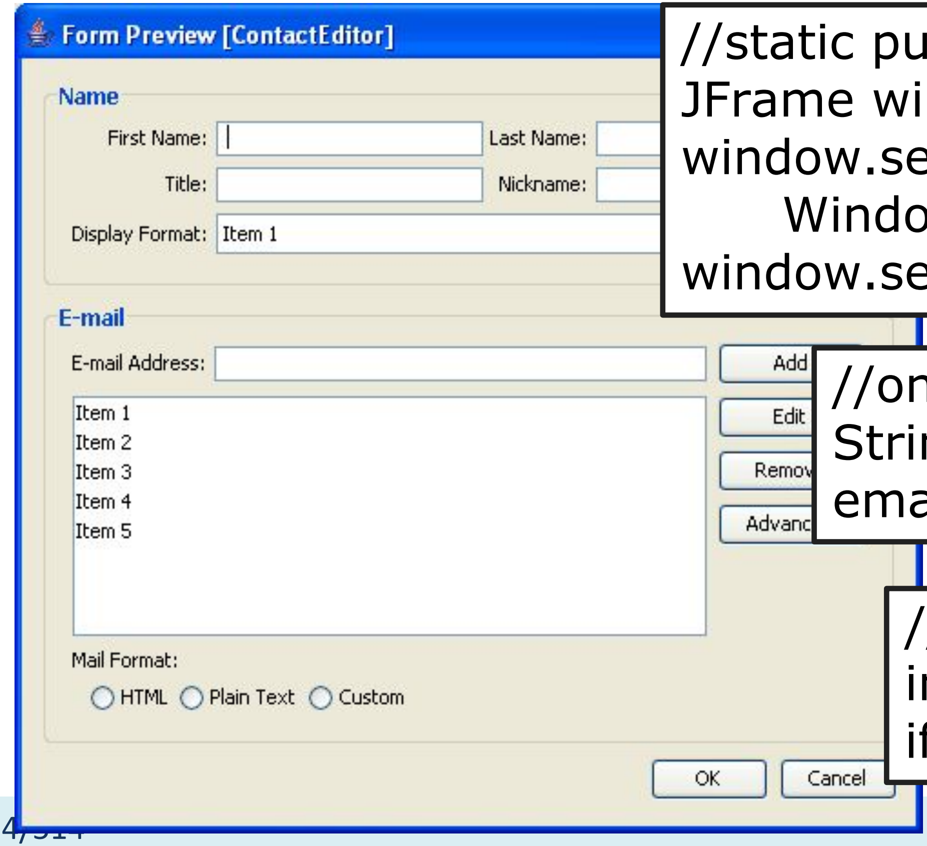


An event-based GUI with a GUI framework

- Setup phase
  - Describe how the GUI window should look
  - Register observers to handle events
- Execution
  - Framework gets events from OS, processes events
    - Your code is mostly just event handlers



Event-based GUIs



```
//static public void main…
JFrame window = …
window.setDefaultCloseOperation(
      WindowConstants.EXIT_ON_CLOSE);
window.setVisible(true);
```

```
//on add-button click:
String email = emailField.getText();
emaillist.add(email);
```

```
//on remove-button click:
int pos =emaillist.getSelectedItem(
if (pos>=0) emaillist.delete(pos);
```

Three Concepts of Importance

- **Thread**: instructions executed in sequence
  - Within a thread, everything happens in order.
  - A thread can start, sleep, and die.
  - You often work on the “main” thread.

Three Concepts of Importance

- **Thread**: instructions executed in sequence
  - Within a thread, everything happens in order.
  - A thread can start, sleep, and die.
  - You often work on the “main” thread.
- **Concurrency**: multiple threads running at the same time
  - Not necessarily *executing* in parallel

Three Concepts of Importance

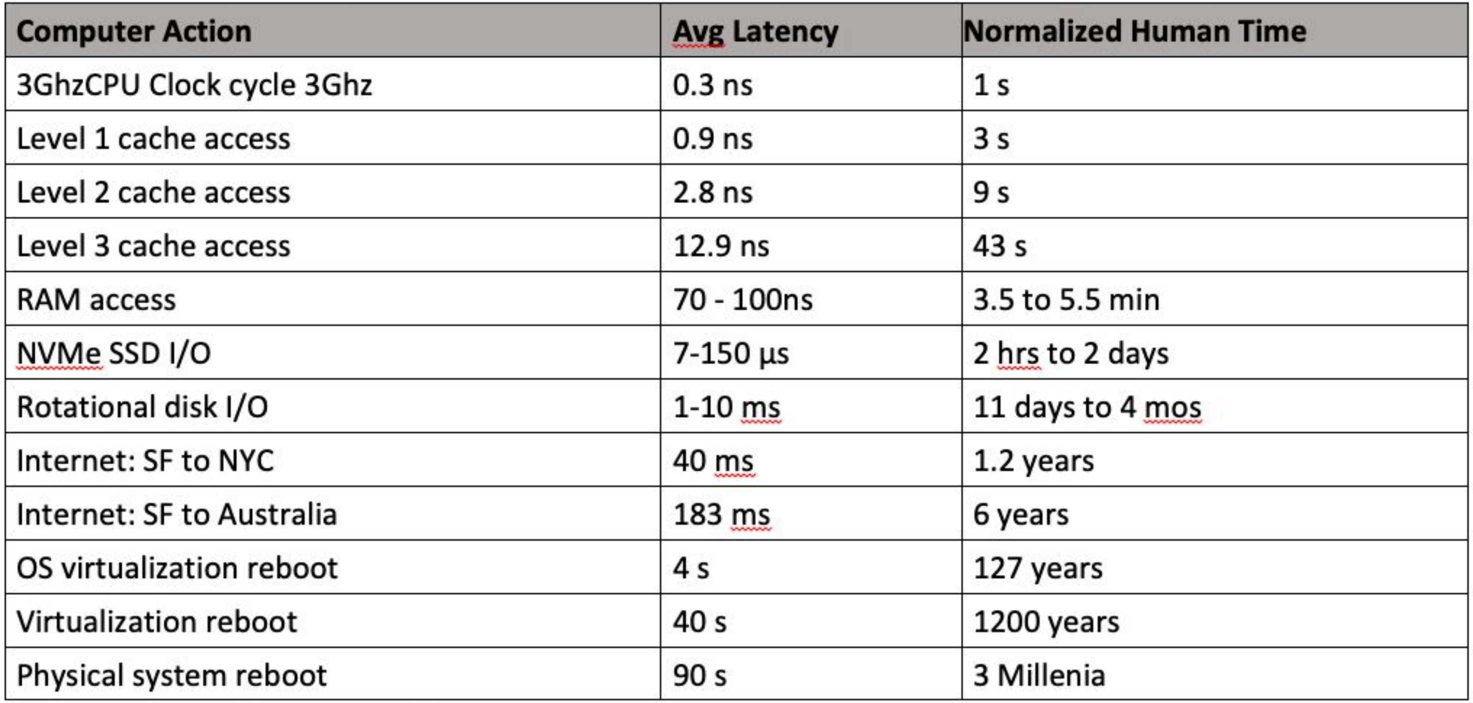
- **Thread**: instructions executed in sequence
  - Within a thread, everything happens in order.
  - A thread can start, sleep, and die.
  - You often work on the “main” thread.
- **Concurrency**: multiple threads running at the same time
  - Not necessarily *executing* in parallel
- **Asynchrony**: computation happening outside the main flow

Where do we want concurrency?

- User interfaces
    - Events can arrive any time
- File I/O
    - Offload work to disk/network/... handler
- Background work
    - Periodically run garbage collection, check health of service
- High-performance computing
    - Facilitate parallelism and distributed computing

Concurrency with file I/O

Key chart:



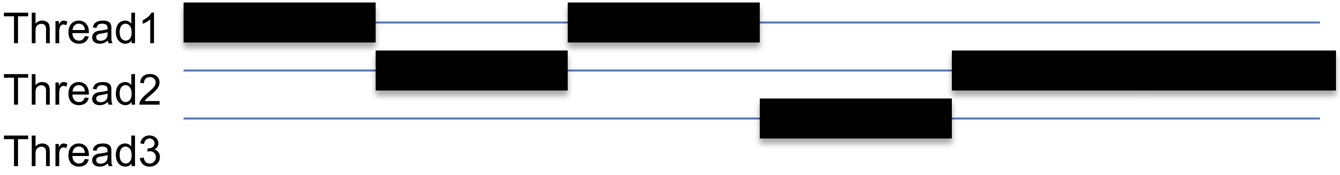
Computer Action	Avg Latency	Normalized Human Time
3GhzCPU Clock cycle 3Ghz	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
RAM access	70 - 100ns	3.5 to 5.5 min
NVMe SSD I/O	7-150 µs	2 hrs to 2 days
Rotational disk I/O	1-10 ms	11 days to 4 mos
Internet: SF to NYC	40 ms	1.2 years
Internet: SF to Australia	183 ms	6 years
OS virtualization reboot	4 s	127 years
Virtualization reboot	40 s	1200 years
Physical system reboot	90 s	3 Millenia

Table 1: Computer Time in Human Terms [i]

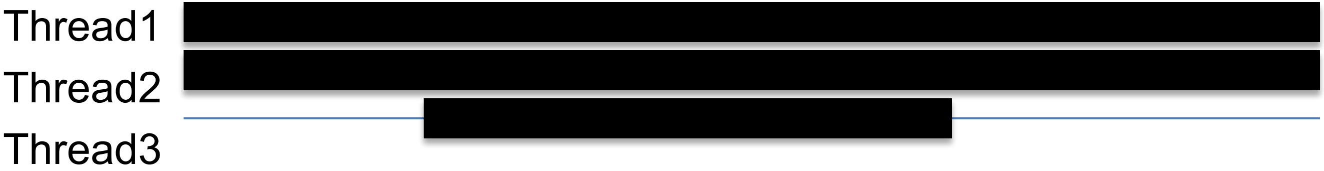
https://formulusblack.com/blog/compute-performance-distance-of-data-as-a-measure-of-latency/

Aside: Concurrency vs. parallelism

- Concurrency without parallelism:



- Concurrency with parallelism:



What is a thread?

- Short for *thread of execution*
- Multiple threads can run in the same program concurrently
- Threads share the same address space
  - Changes made by one thread may be read by others
- Multi-threaded programming
  - Also known as shared-memory multiprocessing

Basic concurrency in Java

- An interface representing a task

```
public interface Runnable {
    void run();
}
```

- A class to execute a task in a thread

```
public class Thread {
    public Thread(Runnable task);
    public void start();
    public void join();
    …
}
```

makes sure that thread is terminated before the next instruction is executed by the program

A simple threads example

```java
public interface Runnable {  // java.lang.Runnable
    public void run();
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);  // Number of threads;

    Runnable greeter = new Runnable() {
        public void run() {
            System.out.println("Hi mom!");
        }
    };
    for (int i = 0; i < n; i++) {
        new Thread(greeter).start();
    }
}
```

A simple threads example

```java
public interface Runnable {  // java.lang.Runnable
    public void run();
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);  // Number of threads;

    Runnable greeter = () -> System.out.println("Hi mom!");
    for (int i = 0; i < n; i++) {
        new Thread(greeter).start();
    }
}
```

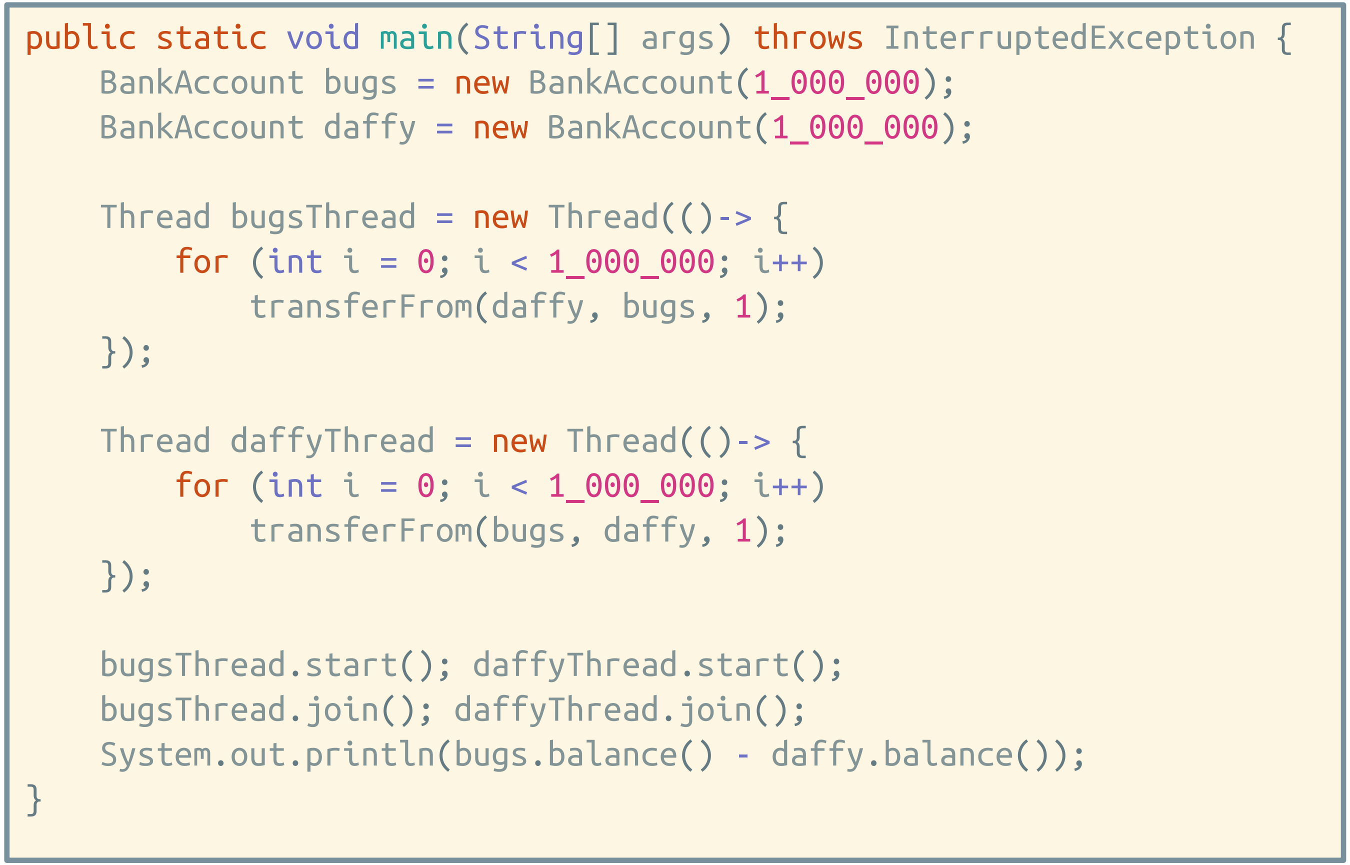
A simple threads example

```java
public interface Runnable {  // java.lang.Runnable
    public void run();
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);  // Number of threads;

    for (int i = 0; i < n; i++) {
        new Thread(() -> System.out.println("Hi mom!")).start();
    }
}
```

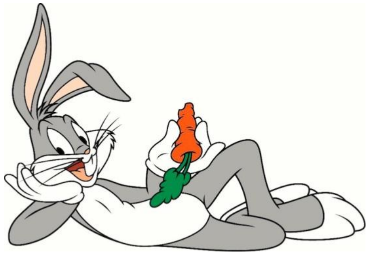
Another example: Money-grab (1)

```
public class BankAccount {
    private long balance;

    public BankAccount(long balance) {
        this.balance = balance;
    }
    static void transferFrom(BankAccount source,
                             BankAccount dest, long amount) {
        source.balance -= amount;
        dest.balance   += amount;
    }
    public long balance() {
        return balance;
    }
}
```

Another example: Money-grab (2)



```
public static void main(String[] args) throws InterruptedException {
    BankAccount bugs = new BankAccount(1_000_000);
    BankAccount daffy = new BankAccount(1_000_000);

    Thread bugsThread = new Thread(()-> {
        for (int i = 0; i < 1_000_000; i++)
            transferFrom(daffy, bugs, 1);
    });

    Thread daffyThread = new Thread(()-> {
        for (int i = 0; i < 1_000_000; i++)
            transferFrom(bugs, daffy, 1);
    });

    bugsThread.start(); daffyThread.start();
    bugsThread.join(); daffyThread.join();
    System.out.println(bugs.balance() - daffy.balance());
}
```



What went wrong?

- Daffy & Bugs threads had a *race condition* for shared data
  - Transfers did not happen in sequence
- Reads and writes interleaved randomly
  - Random results ensued

Safety, Liveness, Performance

CONCURRENCY HAZARDS

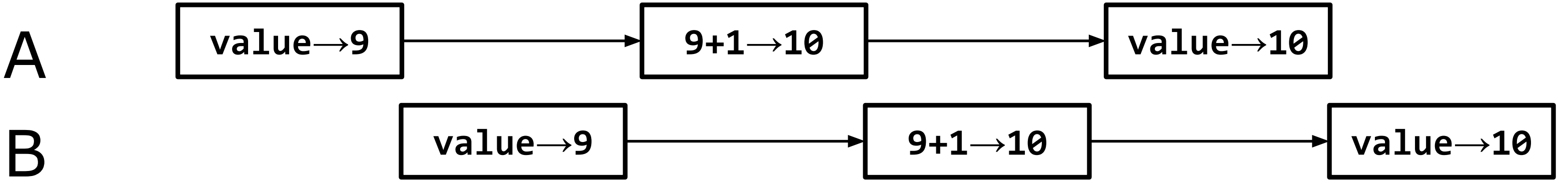
1. Safety Hazard

- The ordering of operations in multiple threads is **unpredictable**.

```java
@NotThreadSafe
public class UnsafeSequence {
    private int value;

    public int getNext() {
        return value++;
    }
}
```

Not atomic

- Unlucky execution of `UnsafeSequence.getNext`



Aside: Atomicity

- An action is *atomic* if it is indivisible
  - Effectively, it happens all at once
    - No effects of the action are visible until it is complete
    - No other actions have an effect during the action
- In Java, integer increment is not atomic

```
i++;
```

is actually

1. Load data from variable `i`
2. Increment data by `1`
3. Store data to variable `i`

Thread Safety

A class is thread safe if it behaves correctly when accessed from multiple threads, regardless of the scheduling or interleaving of the execution of those threads by the runtime environment, and with no additional synchronization or other coordination on the part of the calling code.

2. Liveness Hazard

- Safety: “nothing bad ever happens”
- Liveness: “something good eventually happens”

- Deadlock
  - Infinite loop in sequential programs
  - Thread A waits for a resource that thread B holds exclusively, and B never releases it → A will wait forever
    - E.g., Dining philosophers

- Elusive: depend on relative timing of events in different threads

Deadlock example

Two threads:

A does `transfer(a, b, 10)`   B does `transfer(b, a, 10)`

```
class Account {
  double balance;

  void withdraw(double amount){ balance -= amount; }

  void deposit(double amount){ balance += amount; }

  void transfer(Account from, Account to, double amount){
        synchronized(from) {
            from.withdraw(amount);
            synchronized(to) {
                to.deposit(amount);
            }
        }
  }
}
```

Execution trace:
A: lock a (v)
B: lock b (v)
A: lock b (x)
B: lock a (x)
A: wait
B: wait

Deadlock!

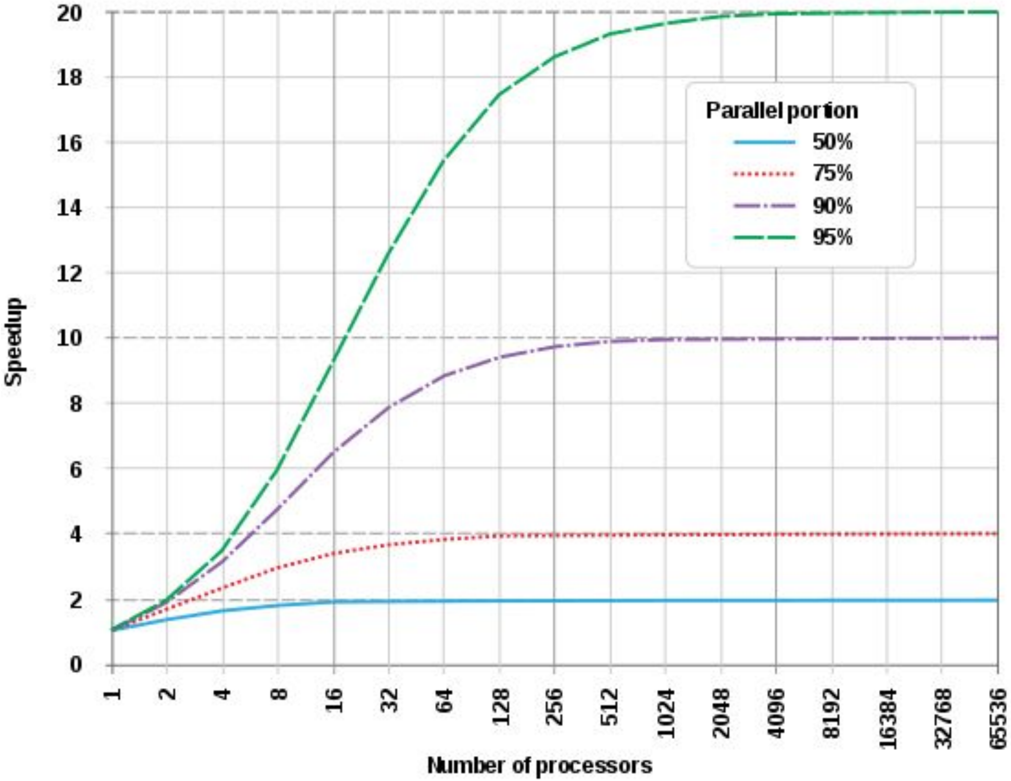
3. Performance Hazard

- Liveness: “something good eventually happens”
- Performance: we want something good to happen quickly

- Multi-threading involves runtime overhead:
  - Coordinating between threads (locking, signaling, memory sync)
  - Context switches
  - Thread creation & teardown
  - Scheduling
- Not all problems can be solved faster with more resources
  - One mother delivers a baby in 9 months

Amdahl’s law

- The speedup is limited by the serial part of the program.



How fast can this run?

- N threads fetch independent tasks from a shared work queue

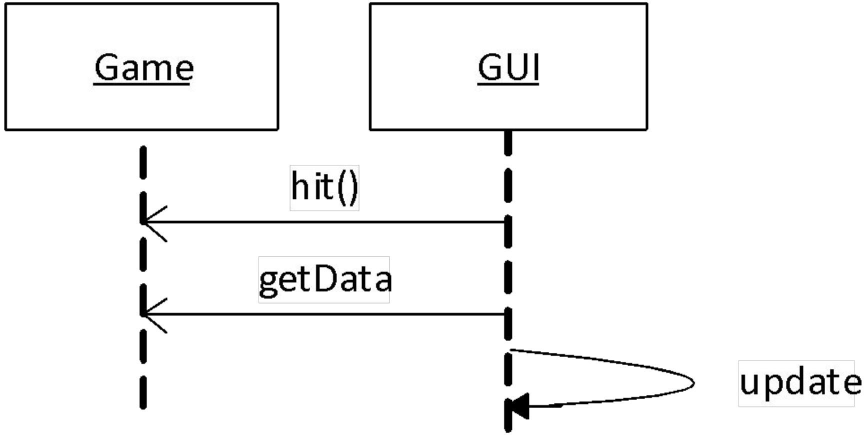
```java
public class WorkerThread extends Thread {
    ...

    public void run() {
        while (true) {
            try {
                Runnable task = queue.take();
                task.run();
            } catch (InterruptedException e) {
                break; /* Allow thread to exit */
            }
        }
    }
}
```

A design challenge

DECOUPLING THE GUI

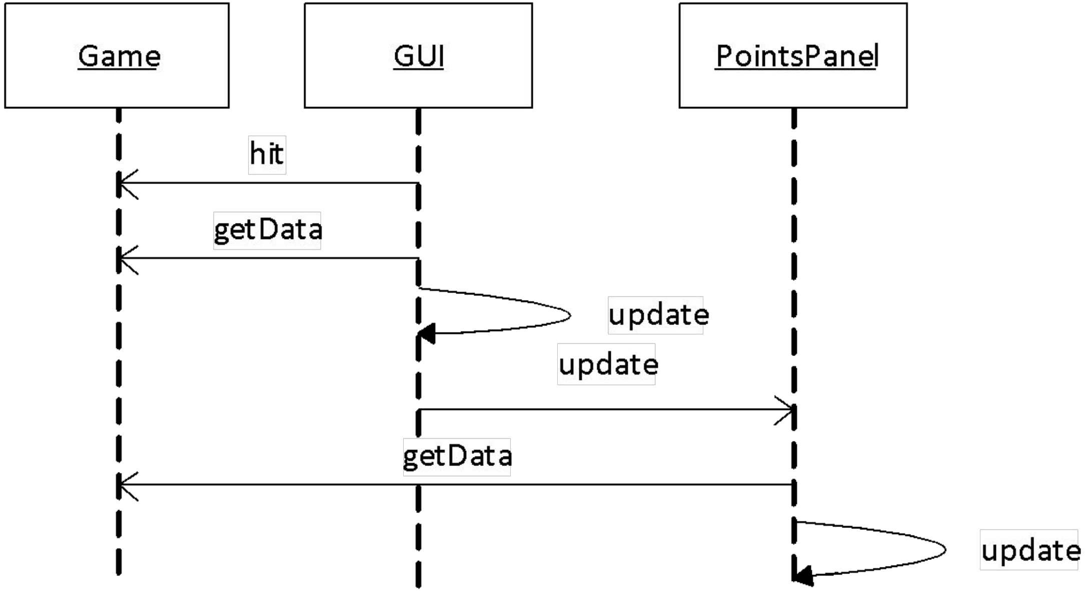
A GUI design challenge

- Consider a blackjack game, implemented by a `Game` class:
  - Player clicks “hit” and expects a new card
  - When should the GUI update the screen?



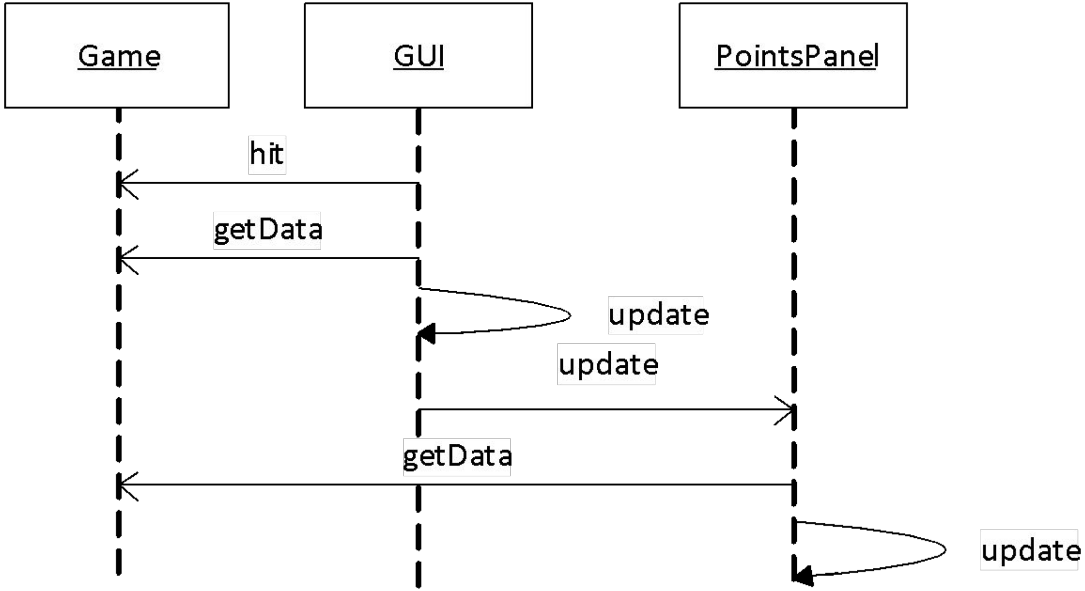
A GUI design challenge, extended

- What if we want to show the points won?



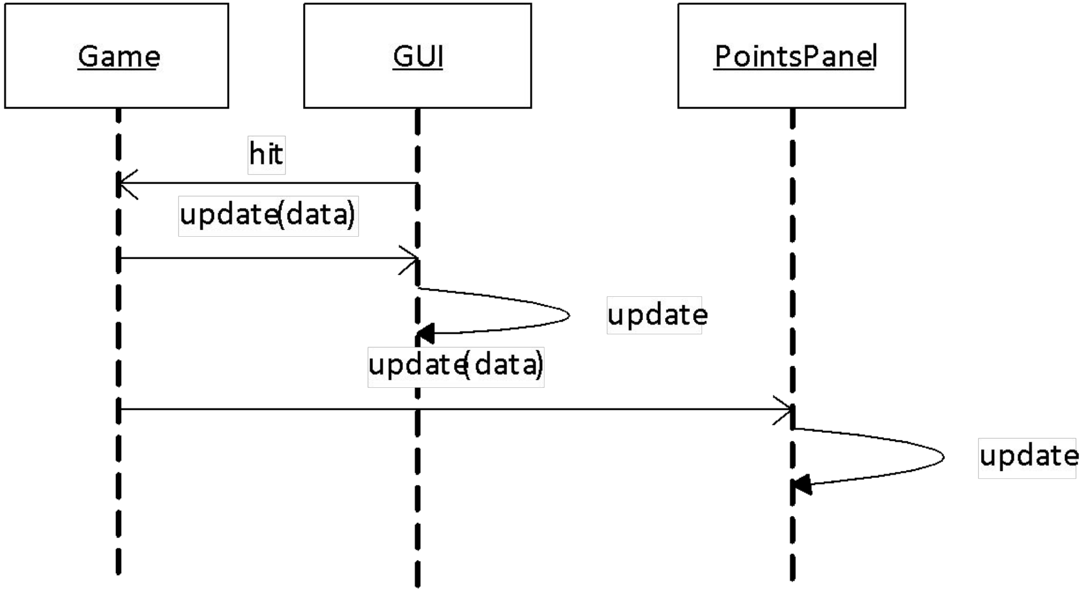
Game updates GUI?

- What if points change for reasons not started by the GUI?
  (or computations take a long time and should not block)



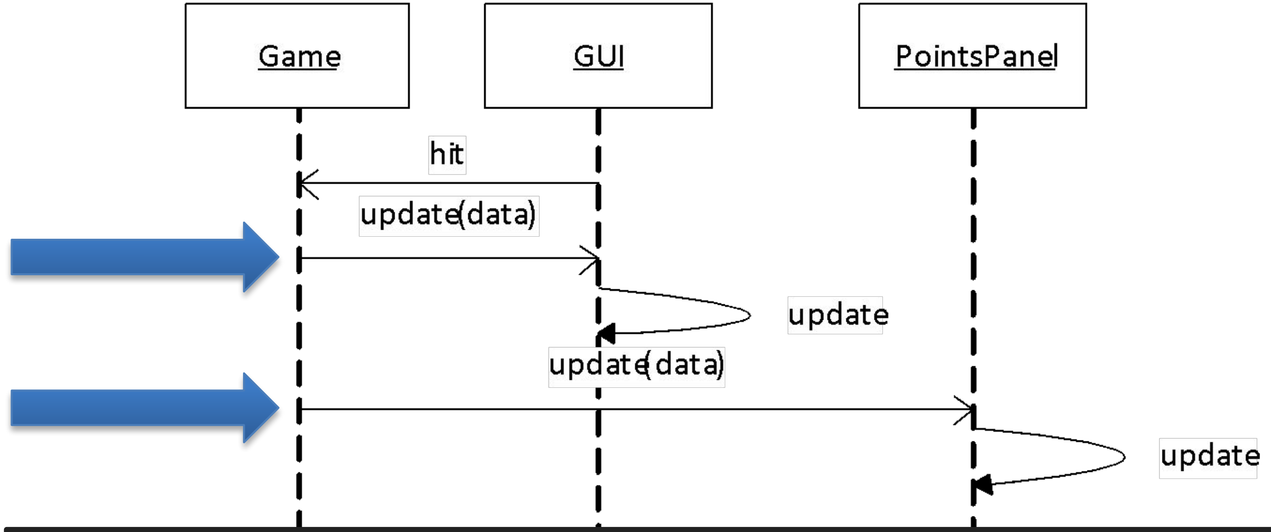
Game updates GUI?

- Let the Game tell the GUI that something happened



Game updates GUI?

- Let the Game tell the GUI that something happened



Problem: This couples the `World` to the GUI implementation.

Core implementation vs. GUI

- Core implementation: Application logic
  - Computing some result, updating data
- GUI
  - Graphical representation of data
  - Source of user interactions
- Design guideline: *Avoid coupling the GUI with core application*
  - Multiple UIs with single core implementation
  - Test core without UI
  - *Design for change, design for reuse, design for division of labor; low coupling, high cohesion*

… to be continued

Designing for Asynchrony & Concurrency

- We are in a new paradigm now
  - We need standardized ways to handle asynchronous and/or concurrent interactions
  - This is how design patterns are born
- A lot of powerful syntax for managing concurrency
  - To be discussed in future classes

Summary

- Thinking past the main loop
  - The world is asynchronous
  - Concurrency helps, in a lot of ways
  - Requires revisiting programming patterns
- Start considering UI design
  - Discussed in more detail next week