Looping Structures
The Plan

While not everyone understands:
1. Motivate loops
2. For loops
3. While loops
4. Do-while loops
5. Equivalence
6. Application of Simulated Collision
7. Practice Problems
Motivation

Why loop?

Sometimes you need to do things again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again, and again...and finally you get tired of typing.
Motivation

Okay, so that's not all. You also loop in order to:

- Group repeatedly executed code for uniformity
- Make the number of repetitions easily changeable
- Repeat events which the number of executions is known only dynamically
- Combine with selection statements to make more complex algorithms
for Loop

```java
for(int i=0; i<10; i++)
{
    System.out.println(i);
}
```
for Loop

```
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}
```
for Loop

for(int i=0; i<10; i++)
{
    System.out.println(i*0.1);
}

Initialization | Condition | Update

for (int i = 0; i < 10; i++)
{
    System.out.println(i * 0.1);
}

Scale factor

Iteration 7
for Loop

```
for(int i=0; i<10; i++)
{
    System.out.println(i*0.1+5);
}
```
while Loop

int i=0;
while (i<10)
{
    System.out.println(i);
    i++;
}

Iteration 9
while Loop

```
int i = 0;
while (i < 10)
{
    System.out.println(i);
    i++;
}
```
while Loop

double i=0;
while(i<1)
{
    System.out.println(i);
    i+=0.1;
}

Why might this be a problem?
do-while Loop

```java
int i=0;
do
{
    System.out.println(i);
i++;
}while(i<=10);
```
do-while Loop

```java
int i = 0;
do {
    System.out.println(i);
    i++;
} while (i <= 10);
```

- **Initialization**: `int i = 0;`
- **Condition**: `while (i <= 10)`
- **Update**: `i++`
- **Flow**: True/False
do-while Loop

```java
int i=0;
do {
    System.out.println(i);
    i++;
} while(i<=10); Notice this semicolon was not here in the while loop!
```
int i=0;
while(i<10)
{
    System.out.println(i);
    i++;  
}

for(int i=0; i<10; i++)
{
    System.out.println(i);  
}
When to use which loop?

Is it known how many times the loop will execute prior to executing the loop body?

Yes → for

No → Is it important for the loop body to always execute at least once?

Yes → do-while

No → while
When to use which loop?

Real answer:

Use which ever structure is most convenient, because all loop structures can be represented as any other loop structure.

Why are there multiple loop structures then?

Simple answer – for the programmer’s convenience.
Application of Simulated Collision

double velocity=3;
double position=1;
double timeStep=0.1;  
//simulate for about 5 seconds
double time=0;
while(time<5)
{
    System.out.println("(\"+time+, \"+position+)\")");
    time+=timeStep;
    position+=velocity*timeStep;
}
Application of Simulated Collision

double velocity0=3;   double velocity1=-2;
double position0=1;   double position1=10;
double timeStep=0.1;
//simulate for about 5 seconds
double time=0;
while(position0!=position1)
{
    System.out.println("p0 is ("+time+", "+position0+")");
    System.out.println("p1 is ("+time+", "+position1+")");
    time+=timeStep;
    position0+=velocity*timeStep;
    position1+=velocity*timeStep;
}

What’s wrong? Why doesn’t the program end?
Application of Simulated Collision

double velocity0=3; double velocity1=-2;
double position0=1; double position1=10;
double timeStep=0.1;
//simulate for about 5 seconds
double time=0;
while(position0<position1) {
    System.out.println("p0 is "+time", "+position0+");
    System.out.println("p1 is "+time", "+position1");
    time+=timeStep;
    position0+=velocity*timeStep;
    position1+=velocity*timeStep;
}
Application of Simulated Collision

double velocity0 = -3;  double velocity1 = 2;
double position0 = 10;  double position1 = 1;
double timeStep = 0.1;
// simulate for about 5 seconds
double time = 0;
while (position0 < position1)
{
    System.out.println("p0 is ("+time+, "+position0+")");
    System.out.println("p1 is ("+time+, "+position1+")");
    time += timeStep;
    position0 += velocity * timeStep;
    position1 += velocity * timeStep;
}

What about now? (notice velocity and position change)
Practice Problems

- Write a loop to print out from 10 to 100 inclusive counting by 10s
- Write a loop that starts with an arbitrary double x and divides it by 2 repeatedly until it is less than 1. Output the number of times the loop executed. What is being computed?
- Write a loop that sums the first x integers where x is a positive integer. Print out the results.
- Write a loop that takes an integer x starting with value 1 and doubles x so long as x is positive. Bonus question: why doesn’t this loop infinitely? Super Bonus question: why does it loop infinitely when x is a double?
Design of Video Game Package

- JFrame
  - Or
  - Applet
- GameLoop
  - Mouse
  - Keyboard
- Animation Canvas
  - Sprite Tracker
  - Sprite Tracker
The Plan

- Basic principles of design
- Motivate why MVC is needed
- Model/View/Controller (MVC) overview
- Model (today)
- View (when we get to Graphical User Interfaces)
- Controller (when we get to Event Handling)
Basic Principles of Design

We want our programs to be

- simple
- functional
- fast (to code and to execute)
- correct
- easy to modify
- easy to extend
- capable of reuse
Why Simple?

- Simple is easy to understand and therefore easier to reuse, modify, and extend.
- Simple has less likelihood for program errors and is therefore more likely to be correct.
- Simple may be faster than complex, certain to code and perhaps in execution time.
Why functional?

- It has to work.
- How well it works is negotiable considering:
  - speed to release
  - cost to release
  - lifetime of code
Why fast?

- Who wants to wait?
- Works better on slower, more out-of-date machines.
- Sometimes crucial to application’s purpose
  - air control
  - nuclear plant facilities
  - weather prediction
  - stock prediction
Why correct?

- In some cases, incorrect execution is unacceptable
  - access control
  - online sales
- In some cases, incorrect execution can be remedied by re-execution
  - operating system locks up, so just reboot
  - database goes down, just restart
- impacts time coding
Why easy to modify?

- Users rarely can tell you exactly what they want
- Even if they could tell you, what users want changes
- Changes in hardware and software mandate code modification (think Y2K)
Why easy to extend?

- Why not make a good thing better?
- Enables multiple releases of functional programs
  - Windows 3.1, 95, 98, 2000, NT, etc.
  - Java 1.0, 1.1, 1.2, 1.3 and now 1.4
- Keep up with increasing competition and demand
Capable of Reuse

- No need to reinvent the wheel
- Easier to build with tools than starting from scratch
- C was used to make C++ compiler, C++ used to make Java, Java used to make Java compiler!
- Reduce, reuse, recycle is good for coding too!
Why we need MVC?

MVC assists in achieving design principles:

- simple idea
- fast to code with a clear ready-made design
- structure makes reuse, modification and extension simple
- MVC design easy to test for correctness
What is MVC?

- **Model**
  - representation of the data
  - has no visual component

- **View**
  - visual display of the data

- **Controller**
  - specifies ways in which the model can be changed
  - between model and view
Simplified 1D Model

- Recall collision detection simulation.
- We’re going to apply MVC to solve the problem more generally.
- We’re going to use Object Oriented Programming
  – what are the objects?
  – what value do they have?
  – how do the values change?
  – how are the values accessible?
What are our objects?

We’re going to model 1D particles and segments.

- **Particle** (similar to the Sprite class)
- **Segment** (similar to extending the Sprite class)
- **Tracker1D** (similar to Tracker)
- **VelocityTracker** (similar to ProjectileTracker)
- **ModelTest** (similar to our JFrame)
class Particle

Attributes
• double position
• Tracker1D

Behavior
• double getPosition()
• void setPosition(double)
• Tracker1D getTracker()
• void setTracker(Tracker1D)
class Segment extends Particle

Attributes

• double position
• double width
• Tracker1D

Behavior

• double getPosition()
• void setPosition(double)
• double getWidth()
• void setWidth(double)
• Tracker1D getTracker()
• void setTracker(Tracker1D)
interface Tracker1D

Attributes
• none

Behavior
• double getPosition()
• void advanceTime(double)
class VelocityTracker

Attributes
- double velocity
- double position

Behavior
- double getPosition()
- void advanceTime(double)
class ModelTest

Attributes
• none

Behavior
• public static void main(String[] args)
More to come....