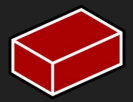
Java101

Crash course in OOP



Redbrick

DCU's Networking Society

Background:

Created by James Goslin in 1995.

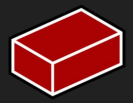
Acquired by Oracle Corporation.

Compiled & Statically typed

Two current LTS versions:

- Java 8
- Java 11

Java != JavaScript



Redbrick
DCU's Networking Society

Where it plays its part.

Four Paradigms of CA:

- Imperative
  - Sequence of commands that update state.
- Declarative
- Functional
- Object-Oriented
  - Defining objects that send messages to each other.
  - Objects have Internal state and Public interfaces.

Redbrick
DCU's Networking Society

What is a Object?

An Object *is a* instantiated Class.

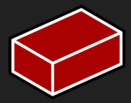
- *Has* state and behaviour

A Class *is a* blueprint or template

- Describes behaviour/state for a object.

Test.java

```java
public class Test {
    int myField = 0;

    public Test(int value) {
        myField = value;
    }
    public int myMethod(int parameter) {
        return parameter + myField;
    }

    public int getMyField(){
        return this.myField;
    }

    Run | Debug
    public static void main(String[] args) {
        Test myTestObject = new Test(5);
        System.out.println(myTestObject.getMyField());
        System.out.println(myTestObject.myMethod(5));
    }

}
```



Redbrick
DCU's Networking Society

Types:

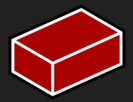
Primitive types (built-ins)

- Eight types
- Boolean, char, int, double

Statically typed

- Forward thinking.
- However Type casting.

```
boolean myBool = true;
char myChar = 'A';
char myOtherChar = '\u0041';
// 8-bit signed integer
byte myByte = 127;
// 16-bit signed integer
short myShort = 32767;
// 32-bit signed integer
int myInt = 2147483647;
// 64-bit signed integer
long myLong = 9223372036854775807L;
// 32-bit
float myFloat = 1.0f;
// 64-bit
double myDouble = 1.0;
```



Redbrick
DCU's Networking Society

Where are **S**trings?!

```
import java.lang.String;
```

Java wants to help you!

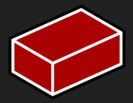
It's a Object.

- Methods available

```
String myString = "Hello World!";
```

Sequence Characters

Char array



Redbrick
DCU's Networking Society

Arrays

A container that holds a fixed number of a single type.

Two approaches:

- Declare and Initialize

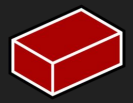
```
char[] myCharArray = {'h','e','l','l','o'};
String[] myStringArray = {"Hello","Redbrick"};
```

- Declare and Assign

```
int[] myIntArray = new int[10];
myIntArray[1] = 5;
```

Note: “new“

```
import java.util.Arrays;
```



Redbrick
DCU’s Networking Society

Method deep dive

Method *is a* function that lives within a Class

They interact with fields.

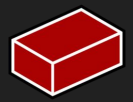
- “this” keyword

Fields and Methods have a access modifiers.

- public, private, protected, package protected

```java
public class BankAccount {

    double balance = 0.0;

    public void deposit(double funds){
        this.balance += funds;
    }

    public void withdraw(double funds){
        this.balance -= funds;
    }

    public double getBalance(){
        return this.balance;
    }
```

```java
public static void main(String[] args) {
    BankAccount myAccount = new BankAccount();
    myAccount.deposit(110.0);
    myAccount.withdraw(60);
    myAccount.getBalance();
}
```



Redbrick
DCU’s Networking Society

Generics

Class or Method that can take many Types.

- Denoted by <T>
- Reusable code

```
public <T> void output(T param){
    System.out.println(param);
    }
```

Redbrick
DCU's Networking Society

Lists

Built in Class that uses Generics.

- Part of Collections Framework
  - Specifically List Interface

```
List <Integer> myIntList = new ArrayList<Integer>();
```

Since its a class…

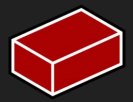
```
myIntList.size()
```

```
myIntList.add(0, 32);
```

There are methods

```
myIntList.add(2);
```

Note: “Integer” is a class.



Redbrick
DCU’s Networking Society

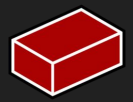
Collections Framework

Provides easy access to data structures.

- Eight Interfaces:
  - Collection Interface
  - List Interface
  - Set
  - Sorted Set
  - Map
  - Map.Entry
  - SortedMap
  - Enumeration

Example:

- LinkedList
- Trees
- HashSet
- Dictionary
- Stack



Redbrick
DCU's Networking Society

Core Concepts of OOP

Inheritance

- Derive a class from another class

Polymorphism

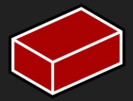
- Method overloading

Encapsulation

- Access Modifiers Getters and Setters, Similar ideas lumped together.

Abstraction

- Only need to know methods not how they work (interface/abstract class)



Redbrick
DCU's Networking Society

Thinking ahead...

Imperative

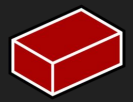
- “And then this happens”

Object Oriented

- “And this will happen”

Difference?

- Imperative is sequential and now
- OOP is planning



Redbrick
DCU’s Networking Society

Thinking ahead P2

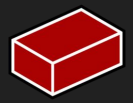
Why two slides?

Have you ever written a program where you have repeated code?

OOP should be:

- ReUsable
- Extendable
- Modular

Back to Concepts!



Redbrick
DCU's Networking Society

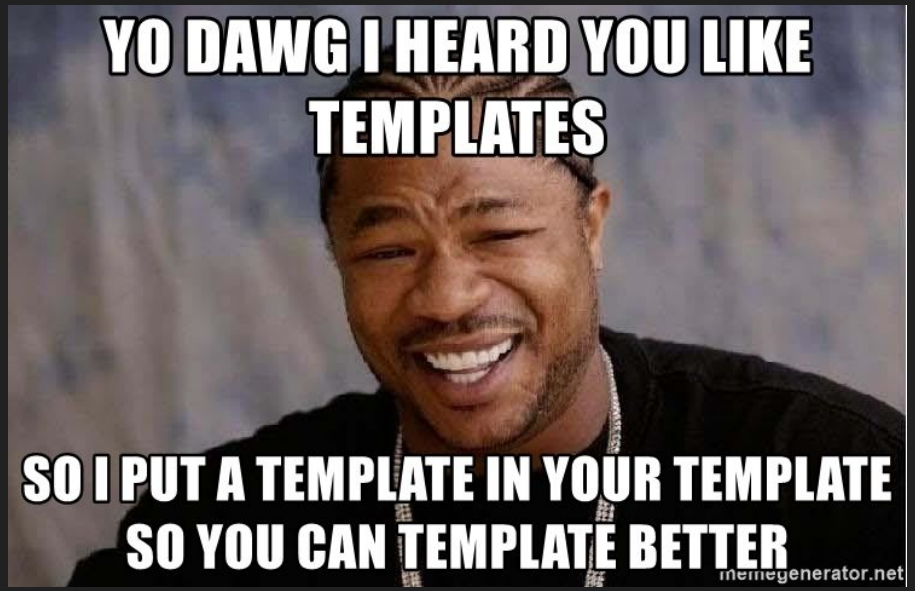
Inheritance

We want to make a zoo…

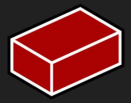
We need animals for our zoo

- Tiger, Wolf, Elephant

Very boring zoo...



Redbrick
DCU's Networking Society



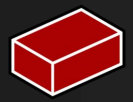
Redbrick
DCU's Networking Society

```java
public class Elephant {

    String name;
    int eyes = 2;
    int legs = 4;
    int tusks = 2;
    boolean hasBackBone = true;
    boolean hasLungs = true;
    boolean hasFangs = false;
    boolean hasClaws = false;
    double velocity =0.0;

    public Elephant( String name) {
        this.name = name;
    }

    public void changeSpeed(double currVel){
        this.velocity = currVel;
    }

    Run | Debug
    public static void main(String[] args) {
        Elephant myElephant = new Elephant("Dumbo");
        myElephant.changeSpeed(10);
    }
}
```

```java
public class Wolf {

    String name;
    int eyes = 2;
    int legs = 4;
    boolean hasBackBone = true;
    boolean hasLungs = true;
    boolean hasFangs = true;
    boolean hasClaws = true;
    double velocity =0.0;

    public Wolf( String name) {
        this.name = name;
    }

    public void changeSpeed(double currVel){
        this.velocity = currVel;
    }

    Run | Debug
    public static void main(String[] args) {
        Wolf myWolf = new Wolf("Ghost");
        myWolf.changeSpeed(10);
    }
}
```

```java
public class Tiger {

    String name;
    int eyes = 2;
    int legs = 4;
    boolean hasBackBone = true;
    boolean hasLungs = true;
    boolean hasFangs = true;
    boolean hasClaws = true;
    double velocity =0.0;

    public Tiger( String name) {
        this.name = name;
    }

    public void changeSpeed(double currVel){
        this.velocity = currVel;
    }

    Run | Debug
    public static void main(String[] args) {
        Tiger myTiger = new Tiger("Tony");
        myTiger.changeSpeed(10);
    }
}
```

```java
public class Mammal {

    String name;
    int eyes = 2;
    int legs = 4;
    boolean hasBackBone = true;
    boolean hasLungs = true;
    double velocity =0.0;

    public Mammal(String name) {
        super();
        this.name = name;
    }
    public void changeSpeed(double currVel){
        this.velocity = currVel;
    }
}
```



Redbrick
DCU's Networking Society

```java
public class Wolf extends Mammal{

    boolean hasFangs = true;
    boolean hasClaws = true;

    public Wolf( String name) {
        super(name);
    }
    public static void main(String[] args) {
        Wolf myWolf = new Wolf("Ghost");
        myWolf.changeSpeed(10);
    }
}
```

```java
public class Wolf extends Mammal{

    boolean hasFangs = true;
    boolean hasClaws = true;

    public Wolf( String name) {
        super(name);
    }
    public static void main(String[] args) {
        Wolf myWolf = new Wolf("Ghost");
        myWolf.changeSpeed(10);
    }
}
```

```java
public class Tiger extends Mammal {

    boolean hasFangs = true;
    boolean hasClaws = true;

    public Tiger( String name) {
        super(name);
    }

    Run | Debug
    public static void main(String[] args) {
        Tiger myTiger = new Tiger("Tony");
        myTiger.changeSpeed(10);
    }
}
```

Polymorphism

Super Class *has a*:

- Method or a field

Sub-Class has the same method or field…

What happens?

Note: Method Overloading

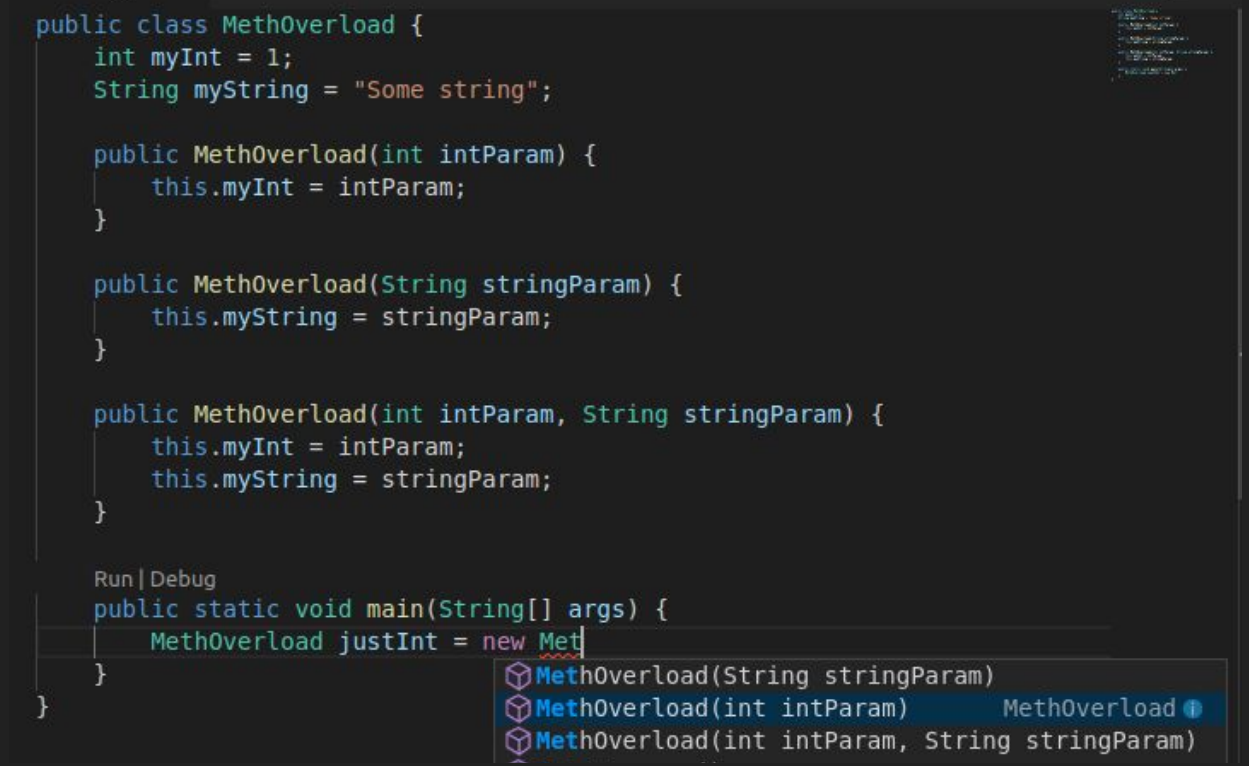
- Can occur in same class

```
class Bike {

    int speed = 60;

    void run(){
        System.out.println("A normal bike");
    }
}
class SportBike extends Bike{

    int speed = 120;

    void run(){
        System.out.println("A fast bike");
    }
    Run | Debug
    public static void main(String[] args) {
        Bike myBike = new SportBike();
        myBike.run();

        System.out.println(myBike.speed);
    }
}
```

SideNote: Method Overloading

Many variations of a single method.

Note: Overloading constructor.



Redbrick
DCU's Networking Society

Encapsulation

Securing data within classes

This entails:

- Access Modifiers
  - public
  - private
  - protected
  - Package protected
- Setters and Getters
  - Used for accessing private data

```java
public class Encapsulation {

    private int myInt;

    public Encapsulation(int intParam) {
        this.myInt = intParam;
    }

    public void setMyInt(int n){
        this.myInt = n;
    }

    public int getMyInt(){
        return this.myInt;
    }
}
```

Encapsulation.java EncapMain.java

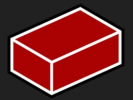
```java
public class EncapMain {

Run | Debug
public static void main(String[] args) {
        Encapsulation myObj = new Encapsulation(5);
        System.out.println(myObj.myInt);
        System.out.println(myObj.getMyInt());
    }
}
```

Redbrick
DCU's Networking Society

```java
public class Encapsulation {

    private int myInt;

    public Encapsulation(int intParam) {
        this.myInt = intParam;
    }

    public void setMyInt(int n){
        this.myInt = n;
    }

    public int getMyInt(){
        return this.myInt;
    }
}
```

Encapsulation.java EncapMain.java

```java
public class EncapMain {

Run | Debug
public static void main(String[] args) {
        Encapsulation myObj = new Encapsulation(5);
        System.out.println(myObj.myInt);
        System.out.println(myObj.getMyInt());
    }
}
```



Redbrick
DCU's Networking Society

Abstraction

Don’t care how a method does something….

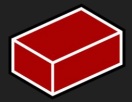
- Once it does it right!
- Example:
  - len()
  - print()

```java
public interface BankInterface {
    public double deposit(double amount);
    public double withdraw(double amount);
}
```

Enforced by Interfaces & Abstract classes.

- Thinking ahead! again...

```java
abstract class BankAbstract {
    abstract double deposit(double amount);
    abstract double withdraw(double amount);
}
```



Redbrick
DCU’s Networking Society

```java
public interface BankInterface {
    public double deposit(double amount);
    public double withdraw(double amount);
}
```

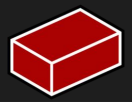
```java
abstract class BankAbstract {
    abstract double deposit(double amount);
    abstract double withdraw(double amount);
}
```

Interface vs Abstract class

- All methods defined in an interface must be implemented.
- This is not the case for Abstract classes

Both allow to scaffold classes!

```java
public class BankAccount implements BankInterface {
    public double balance = 0.0;

    @Override
    public double deposit(double amount) {
        this.balance += amount;
        return this.balance;
    }

    @Override
    public double withdraw(double amount) {
        this.balance -= amount;
        return this.balance;
    }
}
```

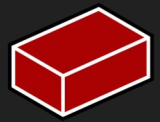


Redbrick
DCU's Networking Society

Questions?!

Covered a lot

We can back pedal :)



Redbrick

DCU's Networking Society