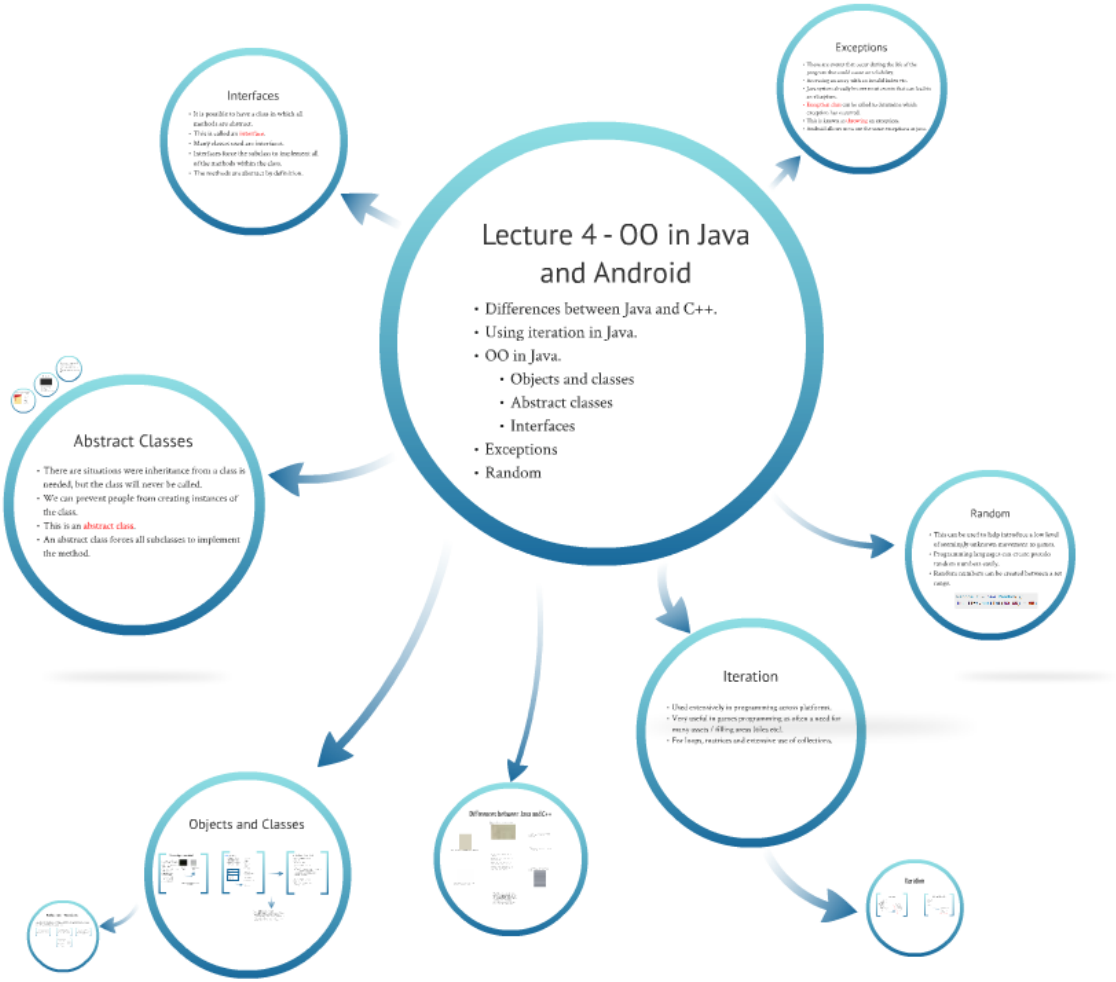
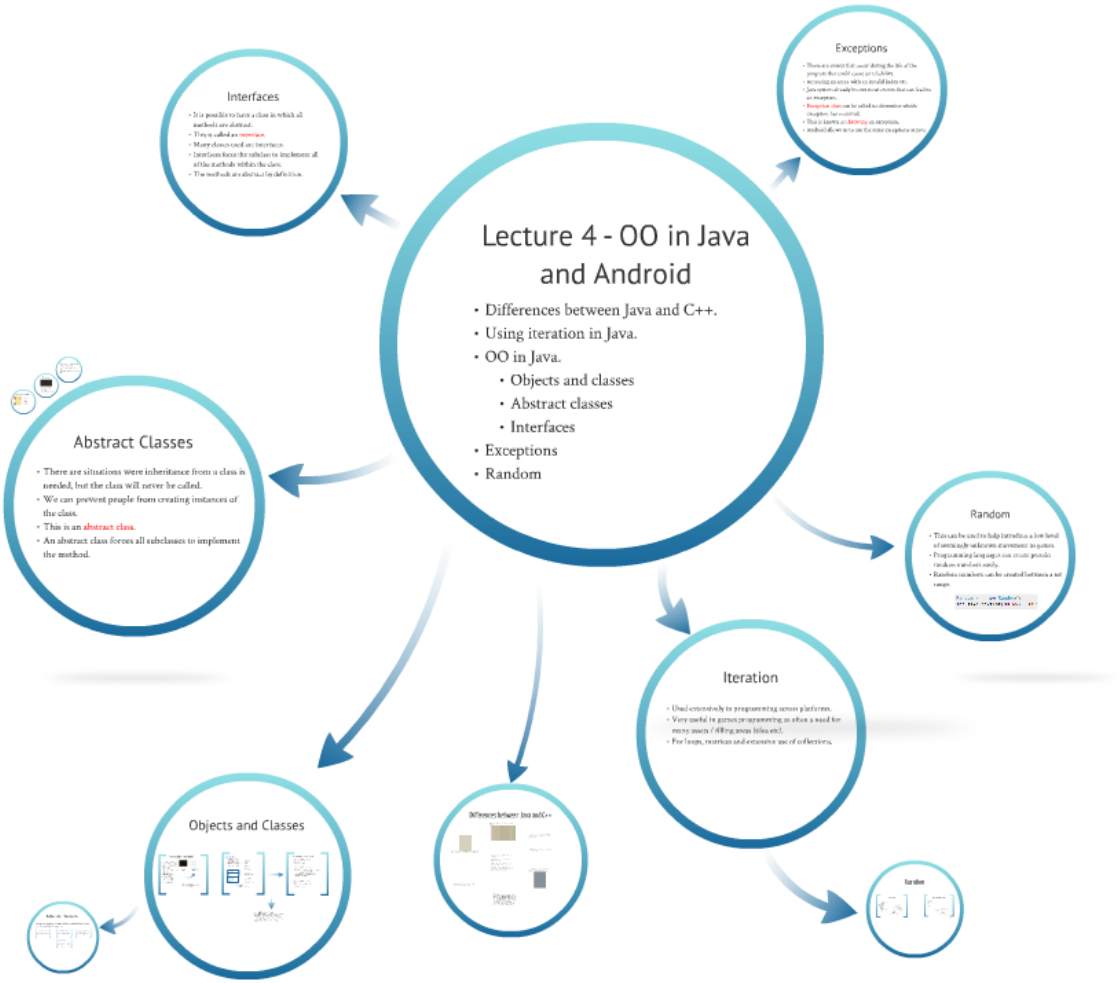
Mobile Games Development 2015/16 - Lecture 4

Dr Jethro Shell
jethros_at_dmu.ac.uk



Mobile Games Development 2015/16 - Lecture 4

Dr Jethro Shell
jethros_at_dmu.ac.uk



Lecture 4 - OO in Java and Android

- Differences between Java and C++.
- Using iteration in Java.
- OO in Java.
  - Objects and classes
  - Abstract classes
  - Interfaces
- Exceptions
- Random

Differences between Java and C++

Java supports classes, but does not support **structures** or **unions**.

Java does not support typedefs, defines, or a preprocessor.

There are no stand-alone functions in Java. Instead, there are only functions that are members of a class, usually called methods.

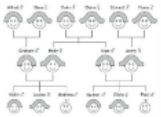
Global functions and global data are not available in Java. There are static variables.

Java uses private, public, and protected but they are different to C++.

private: modifier specifies that the member can only be accessed in its own class.

public: class is visible to all classes everywhere. No modifier, it is visible only within its own package.

private: modifier specifies that the member can only be accessed within its own package and, in addition, by a subclass of its class in another package.



Java does not support multiple inheritance, although **interfaces** can provide some of this process .

There are no destructors in Java. Unused memory is returned to the operating system by way of a garbage collector

- There is no unsigned integer type in Java.
- Java does not support operator overloading, it does however support function overloading.
- Multi-threading is an **easy** implementable part of Java (used a lot in Android games).
- Java contains many data structures by default; vector, stack and dictionary are very useful.

Java does not support typedefs, defines, or a preprocessor.

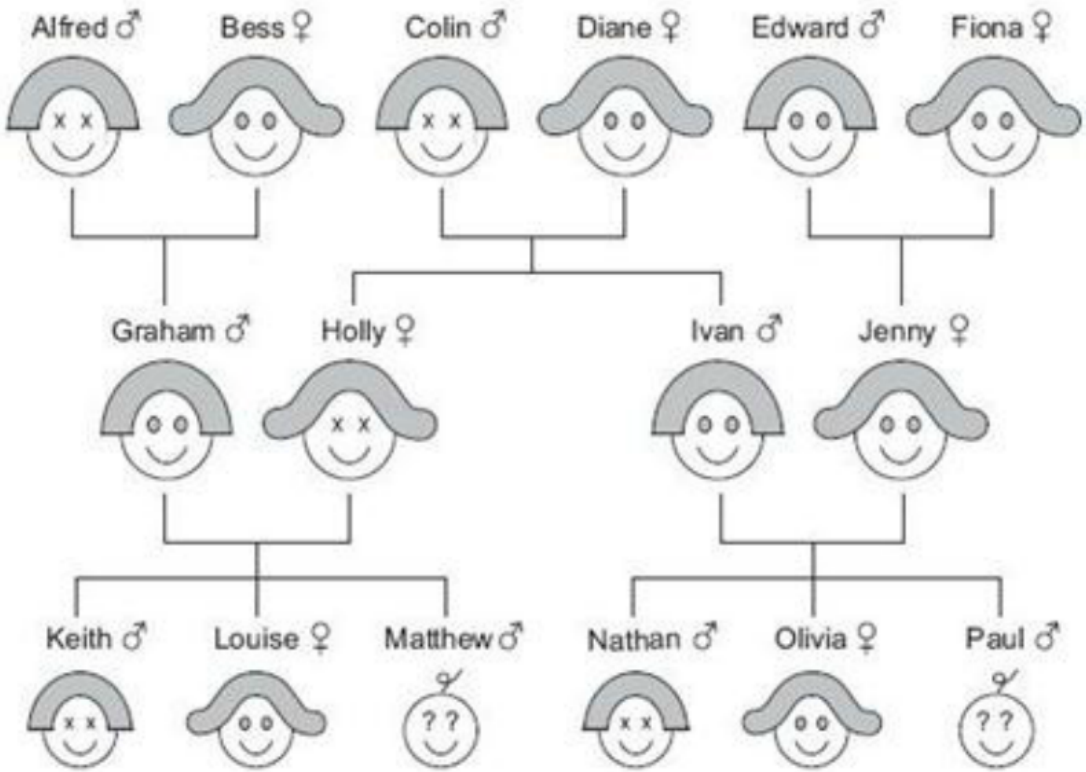
Java supports classes, but does not support **structures** or **unions**.

There are no stand-alone functions in Java. Instead, there are only functions that are members of a class, usually called methods.

Global functions and global data are not available in Java. There are static variables.

There are no destructors in Java. Unused memory is returned to the operating system by way of a garbage collector

- There is no unsigned integer type in Java.
- Java does not support operator overloading, it does however support function overloading.
- Multi-threading is an **easy** implementable part of Java (used a lot in Android games).
- Java contains many data structures by default; vector, stack and dictionary are very useful.



Java does not support multiple inheritance, although **interfaces** can provide some of this process .

Java uses private, public, and protected but they are different to C++.

private: modifier specifies that the member can only be accessed in its own class.

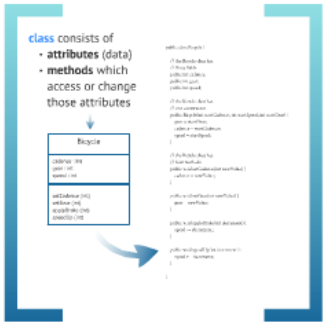
public: class is visible to all classes everywhere. No modifier, it is visible only within its own package.

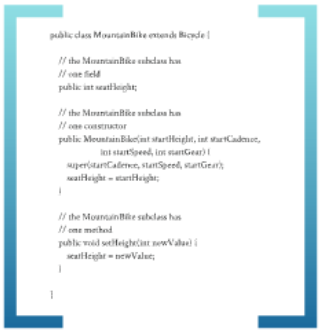
private: modifier specifies that the member can only be accessed within its own package and, in addition, by a subclass of its class in another package.

Lecture 4 - OO in Java and Android

- Differences between Java and C++.
- Using iteration in Java.
- OO in Java.
  - Objects and classes
  - Abstract classes
  - Interfaces
- Exceptions
- Random

Objects and Classes





```
public class MountainBike extends Bicycle {

    // the MountainBike subclass has
    // one field
    public int seatHeight;

    // the MountainBike subclass has
    // one constructor
    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass has
    // one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

}
```

- MountainBike is a subclass of Bicycle.
- MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it.
- **Private Members** - Does not inherit, accessed by methods
- **Casting Objects** - MountainBike is descended from Bicycle and Object, so can be used in place of those items.

What is object - orientation?

- As computers have become more powerful, so software has become more complex to deal with them.
- Demands on developers have become greater.
- Large, unstructured programs become increasingly difficult to develop. A **modular** approach was needed.

IBM XT - 1981
- 4-6MHz
- 256KB RAM

Nexus 5 - 2013
- Quad-core 2.3 GHz
- 2GB RAM

Can consider component based approach?

- As computers have become more powerful, so software has become more complex to deal with them.
- Demands on developers have become greater.
- Large, unstructured programs become increasingly difficult to develop. A **modular** approach was needed.

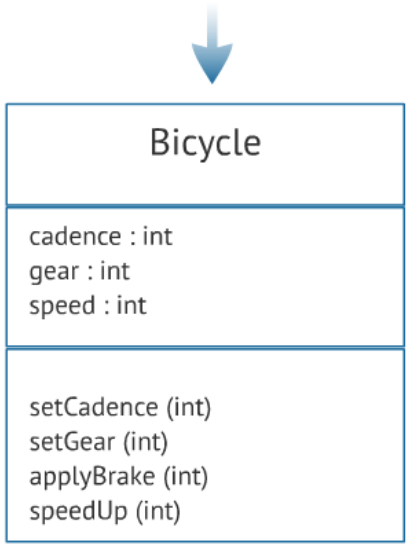
IBM XT - 1981
- 4-6MHz
- 256KB RAM

Can cons

Can consider component based approach?

class consists of

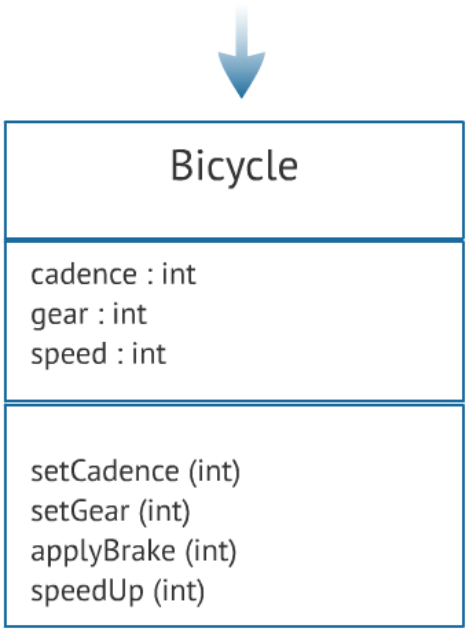
- **attributes** (data)
- **methods** which access or change those attributes



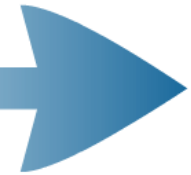

```
public class Bicycle {

   // the Bicycle class has
   // three fields
   public int cadence;
   public int gear;
   public int speed;

   // the Bicycle class has
   // one constructor
   public Bicycle(int startCadence, int startSpeed, int startGear) {
      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
   }

   // the Bicycle class has
   // four methods
   public void setCadence(int newValue) {
      cadence = newValue;
   }

   public void setGear(int newValue) {
      gear = newValue;
   }

   public void applyBrake(int decrement) {
      speed -= decrement;
   }

   public void speedUp(int increment) {
      speed += increment;
   }

}
```

ass consists of

- **attributes** (data)
- **methods** which access or change those attributes




```
public class Bicycle {

   // the Bicycle class has
   // three fields
   public int cadence;
   public int gear;
   public int speed;

   // the Bicycle class has
   // one constructor
   public Bicycle(int startCadence, int startSpeed, int startGear) {
      gear = startGear;
      cadence = startCadence;
      speed = startSpeed;
   }

   // the Bicycle class has
   // four methods
   public void setCadence(int newValue) {
      cadence = newValue;
   }

   public void setGear(int newValue) {
      gear = newValue;
   }

   public void applyBrake(int decrement) {
      speed -= decrement;
   }

   public void speedUp(int increment) {
      speed += increment;
   }

}
```



```
public class MountainBike extends Bicycle {

    // the MountainBike subclass has
    // one field
    public int seatHeight;

    // the MountainBike subclass has
    // one constructor
    public MountainBike(int startHeight, int startCadence,
                        int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }

    // the MountainBike subclass has
    // one method
    public void setHeight(int newValue) {
        seatHeight = newValue;
    }

}
```

- MountainBike is a subclass of Bicycle.
- MountainBike inherits all the fields and methods of Bicycle and adds the field seatHeight and a method to set it.
- **Private Members** - Does not inherit, accessed by methods
- **Casting Objects** - MountainBike is descended from Bicycle and Object, so can be used in place of those items.

Modifiers: Public, Private and Static

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

Default Access Modifier - No keyword:

A variable or method declared without any access control modifier is available to any other class in the same package

```
String noKeyword = "NoKeyword";
```

Private Access Modifier - private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Public Access Modifier - public:

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

Default Access Modifier - No keyword:

A variable or method declared without any access control modifier is available to any other class in the same package

```
String noKeyword = "NoKeyword";
```

Private Access Modifier - private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

```
public class EnemySpaceCraft extends SpaceCraft{

    private int score;

    @Override
    void draw() {
        /**Draw the ship**/
    }

    public int getScore(){
        return score;
    }

}
```

```java
public class EnemySpaceCraft extends SpaceCraft{

    private int score;

    @Override
    void draw() {
        /**Draw the ship**/
    }

    public int getScore(){
        return score;
    }

}
```

Public Access Modifier - public:

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

Static Variables: The static key word is used to create variables that will exist independently of any instances created for the class. Only one copy of the static variable exists regardless of the number of instances of the class.

Static variables are also known as class variables. Local variables cannot be declared static.

```
public class EnemySpaceCraft extends SpaceCraft{

    private static int score = 0;

    protected static int getScore() {
      return score;
    }

    public static void addScore() {
      score++;
    }

    @Override
    void draw() {
        /**Draw the ship**/
    }

}
```

Static Methods: The static key word is used to create methods that will exist independently of any instances created for the class.

Static methods do not use any instance variables of any object of the class they are defined in. Static methods take all the data from parameters and compute something from those parameters, with no reference to variables.

```java
public class EnemySpaceCraft extends SpaceCraft{

    private static int score = 0;

    protected static int getScore() {
      return score;
    }

    public static void addScore() {
      score++;
    }

    @Override
    void draw() {
        /**Draw the ship**/
    }

}
```

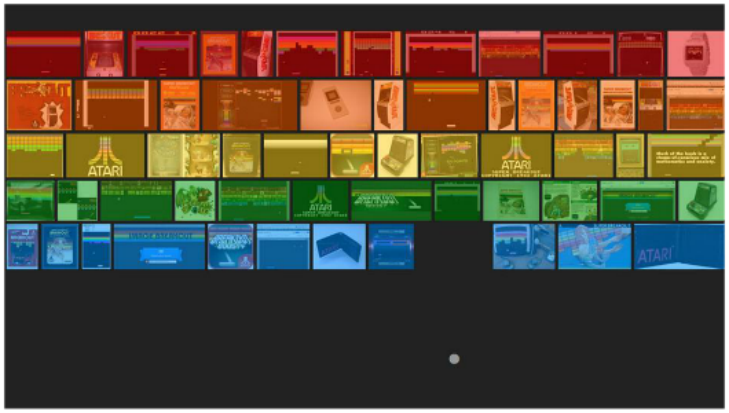
Abstract Classes

- There are situations were inheritance from a class is needed, but the class will never be called.
- We can prevent people from creating instances of the class.
- This is an abstract class.
- An abstract class forces all subclasses to implement the method.

Retro Space Commanders

- Location
- Thrust
- Size
- Velocity
- Position
- Weapons

Breakout



- Need a class for the blocks.
- Different types of blocks.
- Different attributes.
- Different blocks, similar functions.

How is it constructed?

- Uses the abstract class definition.
- Primitives can be defined within the class.
- Unlike an interface, not all methods need to be used.
- Defining methods as abstract can force those to be used.

```
abstract class Spacecraft {
    private int width;
    private int height;
    private int velocity;
    private int health;

    public Spacecraft(int widthIn, int heightIn, int velocityIn, int healthIn){
        width = widthIn;
        height = heightIn;
        velocity = velocityIn;
        health = healthIn;
    }

    public int getWidth(){
        return width;
    }

    public int getHeight(){
        return heigh;
    }

    public int getVelocity(){
        return velocity;
    }

    public int getHealth(){
        return health;
    }

    abstract public int getSize();  //this forces all subclasses of the class
                                    //to have this method
}
```

```
public class EnemySpaceCraft extends SpaceCraft{

    @Override
    void draw() {
        // TODO Auto-generated method stub

    }

}
```

```
abstract class Spacecraft {
    private int width;
    private int height;
    private int velocity;
    private int health;

    public Spacecraft(int widthIn, int heightIn, int velocityIn, int healthIn){
        width = widthIn;
        height = heightIn;
        velocity = velocityIn;
        health = healthIn;
    }

    public int getWidth(){
        return width;
    }

    public int getHeight(){
        return height;
    }

    public int getVelocity(){
        return velocity;
    }

    public int getHealth(){
        return health;
    }

    abstract public int getSize();  //This forces all subclasses of the class
                                    //to have this method
}
```

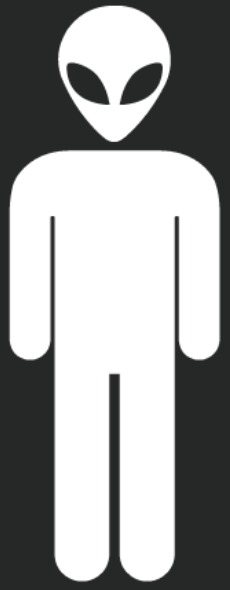
```java
public class EnemySpaceCraft extends SpaceCraft{

    @Override
    void draw() {
        // TODO Auto-generated method stub

    }

}
```

Interfaces

- It is possible to have a class in which all methods are abstract.
- This is called an interface.
- Many classes used are interfaces.
- Interfaces force the subclass to implement all of the methods within the class.
- The methods are abstract by definition.

Interface Class

- In developing a game we may want a framework.
- When a class from the framework is implemented, all methods are used by the subclass.



Implement Alien class, forced to have:

- run()
- fight()
- getStrength()
- setWeapons()

```
interface Weapons {
    public int weaponType (String t); //No body is defined

}
```

- set

```
interface Weapons {
    public int weaponType (String t); //No body is defined

}
```

```java
public class EnemySpacecraft extends Spacecraft implements Weapons{
    private boolean cloakDevice;

    public EnemySpacecraft (int widthIn, int heightIn,
            int velocityIn, int healthIn, boolean cloak){
            super(widthIn,heightIn,velocityIn,healthIn);
            cloakDevice = cloak;
    }

    public boolean getCloakStatus(){
        return cloakDevice;
    }

    public int getSize(){
        return getWidth() * getHeight();
    }

    public int weaponType(String t){
        int powerLevel;
        if (t == "HEAVY"){
            powerLevel = 10;
        }
        else if (t == "MEDIUM"){
            powerLevel = 5;
        }
        else {
            powerLevel = 0;
        }

        return powerLevel;

    }
}
```

Exceptions

- These are events that occur during the life of the program that could cause unreliability.
- Accessing an array with an invalid index etc.
- Java system already knows most events that can lead to an exception.
- Exception class can be called to determine which exception has occurred.
- This is known as throwing an exception.
- Android allows us to use the same exceptions as java.

```java
@Override
public void onDraw(Canvas canvas) {
    try{
        if (blocks.size() > 0){
            for (int i = 0; i < blocks.size(); i++){
                blocks.get(i).drawRect(paint, canvas);
            }
        }
    }catch(Exception e){
        Log.e("Error","Out of scope");
    }

}
```

Random

- This can be used to help introduce a low level of seemingly unknown movement to games.
- Programming languages can create pseudo random numbers easily.
- Random numbers can be created between a set range.

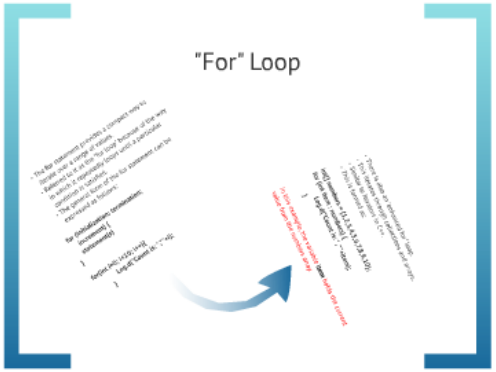
```
Random r = new Random();
int i1=r.nextInt(80-65) + 65;
```

Iteration

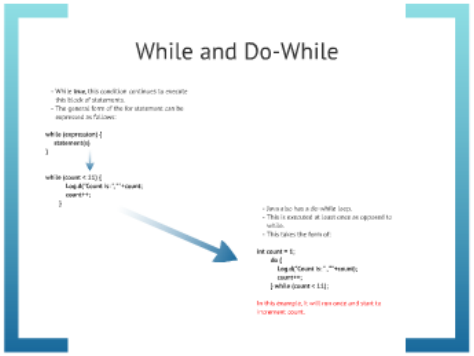
- Used extensively in programming across platforms.
- Very useful in games programming as often a need for many assets / filling areas (tiles etc).
- For loops, matrices and extensive use of collections.

Iteration

"For" Loop



While and Do-While



"For" Loop

- The **for** statement provides a compact way to iterate over a range of values.
- Referred to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied.
- The general form of the for statement can be expressed as follows:

```
for (initialization; termination;
    increment) {
    statement(s)
}
```

```
for(int i=0; i<10; i++){
        Log.d("Count is: " ,""+i);
}
```

- There is also an "enhanced for" loop.
- This iterates through collections and arrays.
- Similar to iterators in C++.
- This is formed as:

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
for (int item : numbers) {
        Log.d("Count is: ", ""+item);
}
```

In this example, the variable **item** holds the current value from the numbers array.

- The **for** statement provides a compact way to iterate over a range of values.
- Referred to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied.
- The general form of the for statement can be expressed as follows:

```
for (initialization; termination;
     increment) {
    statement(s)
}
```

```
for(int i=0; i<10; i++){
            Log.d("Count is: ",""+i);
        }
```

- There is also an "enhanced for" loop.
- This iterates through collections and arrays.
- Similar to iterators in C++.
- This is formed as:

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
for (int item : numbers) {
        Log.d("Count is: ","+item);
    }
```

In this example, the variable **item** holds the current value from the numbers array.

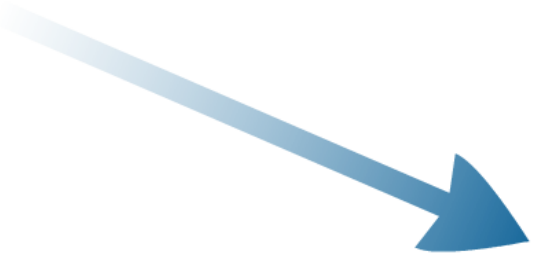
```
static class Foo {
    int mSplat;
}

Foo[] mArray = ...

public void zero() {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one() {
    int sum = 0;
    Foo[] localArray = mArray;
    int len = localArray.length;

    for (int i = 0; i < len; ++i) {
        sum += localArray[i].mSplat;
    }
}

public void two() {
    int sum = 0;
    for (Foo a : mArray) {
        sum += a.mSplat;
    }
}
```

While and Do-While

- While **true**, this condition continues to execute this block of statements.
- The general form of the for statement can be expressed as follows:

```
while (expression) {
    statement(s)
}
```

```
while (count < 11) {
        Log.d("Count is: ", ""+count;
        count++;
    }
```



- Java also has a do-while loop.
- This is executed at least once as opposed to while.
- This takes the form of:

```
int count = 1;
    do {
        Log.d("Count is: ", ""+count);
        count++;
    } while (count < 11);
```

In this example, it will run once and start to increment count.

- While **true**, this condition continues to execute this block of statements.
- The general form of the for statement can be expressed as follows:

```
while (expression) {
    statement(s)
}
```

```
while (count < 11) {
        Log.d("Count is: ",""+count;
        count++;
    }
```

- Java also has a do-while loop.
- This is executed at least once as opposed to while.
- This takes the form of:

```
int count = 1;
        do {
            Log.d("Count is: ",""+count);
            count++;
        } while (count < 11);
```

In this example, it will run once and start to increment count.

Mobile Games Development 2015/16 - Lecture 4

Dr Jethro Shell
jethros_at_dmu.ac.uk