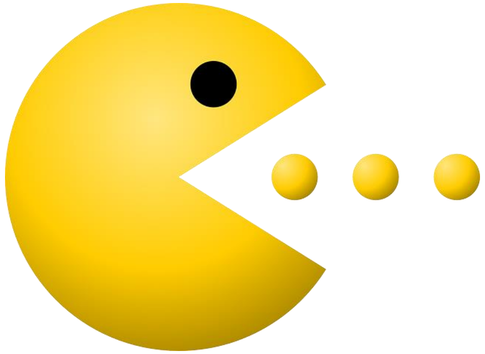
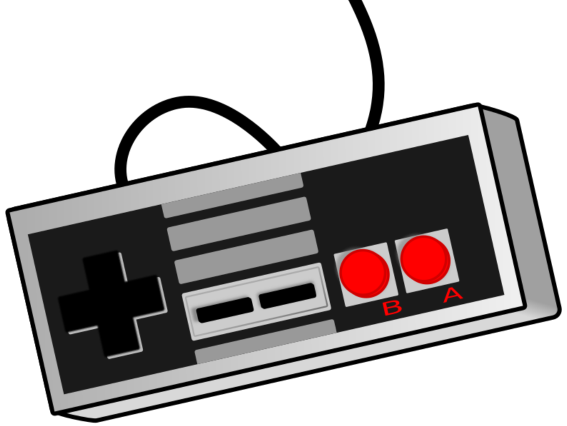




Lecture 01 - Genres and Programming Basics

PREPARED BY NICOLAS BERGERON

420-141-VA - GAME PROGRAMMING 1 - VANIER COLLEGE

What is a Game?

Chess

Monopoly

Pac-man

Half-Life 2

The Sims 2

Dance Dance Revolution

Snakes & Ladders

Billiards

Ping-pong

Poker

Roulette

Professional soccer

Training flight simulator

Dolls

Dating

Treaty negotiation

Business meeting

Poetry course

Cooking

Karaoke

Stock market investing

Tax code

Music concert

Reading a book

Hiking

sleeping

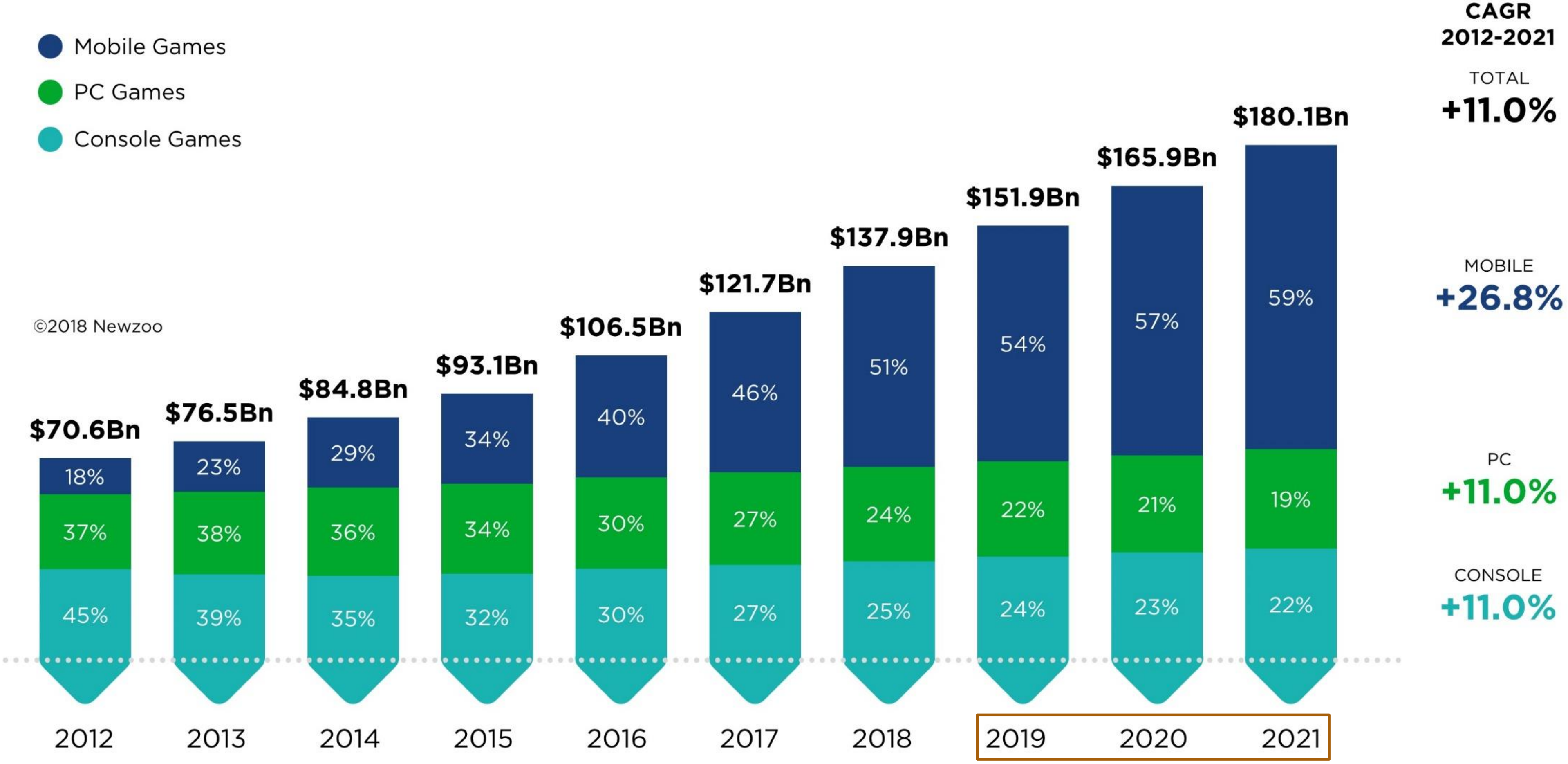
Properties of a Game

- Player(s)
- Goals (say winning/losing)
- Choices that affect the outcome
- Rules
- Consequences of winning or losing that are optional

Games are generally, entertainment activities in which players make choices constrained by rules in pursuit of objective goals that they have a fair chance of achieving.

2012-2021 GLOBAL GAMES MARKET

REVENUES PER SEGMENT 2012-2021 WITH COMPOUND ANNUAL GROWTH RATES



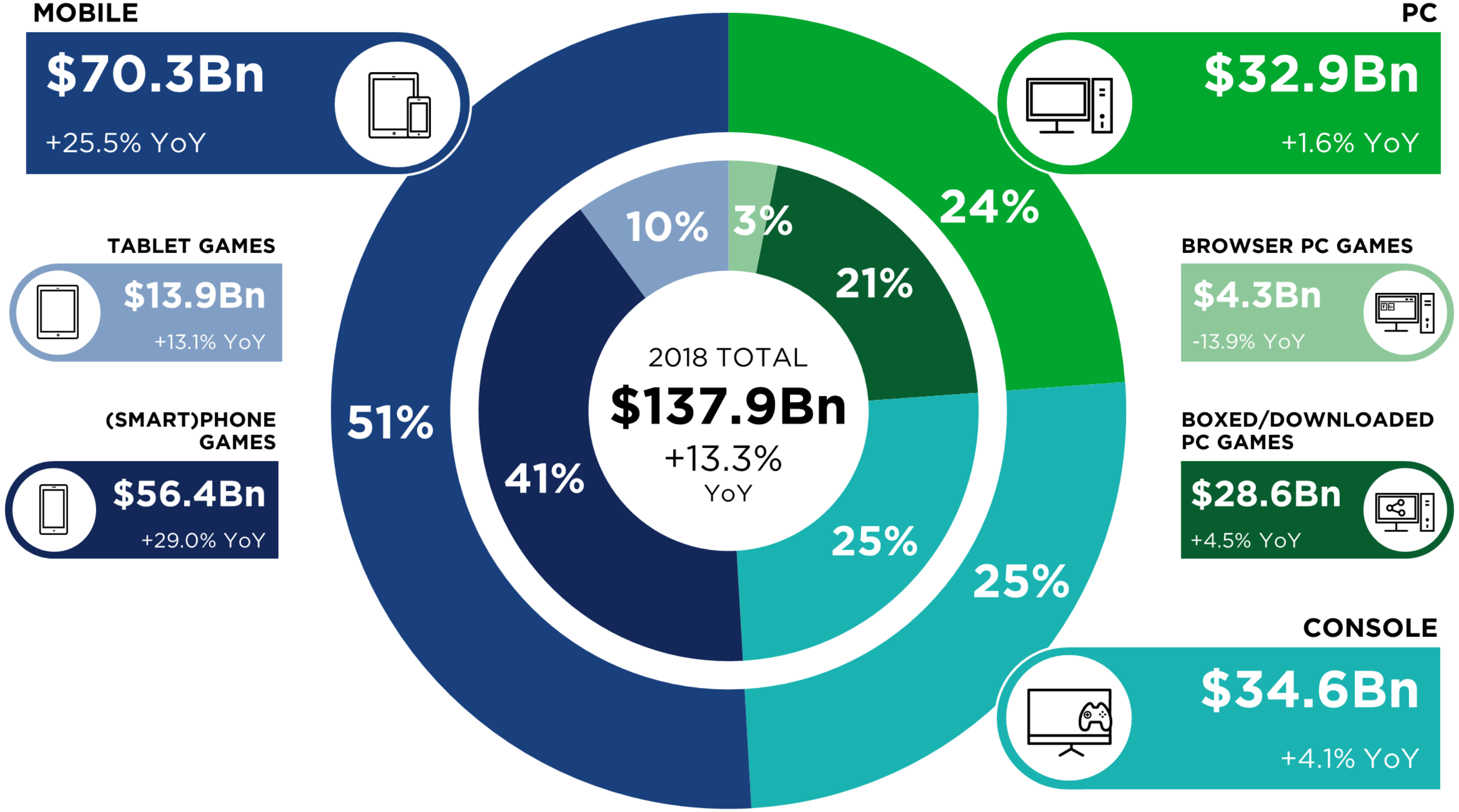
Projected

Source: ©Newzoo | April 2018 Quarterly Update | Global Games Market Report

newzoo.com/globalgamesreport

2018 GLOBAL GAMES MARKET

PER DEVICE & SEGMENT WITH YEAR-ON-YEAR GROWTH RATES



Source: ©Newzoo | April 2018 Quarterly Update | Global Games Market Report

newzoo.com/globalgamesreport

In 2018, mobile games will generate

$70.3Bn

or **51%** of the global market.

newzoo

Game Genres and Characteristics

Adventure Games

Often story-based games with puzzle solving and narration

- Generally has a large, complex world with many interesting characters and a good plot

Generally not real-time games

- Can take as much time as wanted to take a turn; nothing else happens in mean time
- Action-adventure hybrids can be real-time

GRIM FANDANGO
THE SECRET OF MONKEY ISLAND
Day of the Tentacle
MYST
CBM 64
Peter Gerrard
THE CURSE OF MONKEY ISLAND
FULL THROTTLE
THE DIG
INDIANA JONES and the FATE of ATLANTIS

Action Games

Real-time games requiring quick reactions to events

- Includes first-person shooters (FPS) like Quake and Unreal Tournament, platformers like Mario and Sonic, maze games like Pac-Man, and shooters like Space Invaders
- Opponents are either human players or computer

Far less cerebral than adventure games

- Players are looking for fast-paced action
- Still action games include tactical and strategic elements

Role-Playing Games (RPG)

Player directs a hero (or group) on a series of quests

- Huge world with unfolding story
- Players micromanage their characters
- Characters tend to grow in strength and abilities
- Combat generally important to gain experience, money and strength

Fantasy RPGs feature complex magical systems and diverse races of characters.

Strategy Games

Manage a limited set of resources to achieve a predetermined goal

- Resource management : what units to create and how to deploy them
- Trade offs in time, money, and raw materials

Can be either turn-based or real-time

- Turn-based strategies allows thinking and making decisions at your own pace
- Real-time strategies (RTS) have all opponents thinking and acting at the same time with no turns

Opponents can be computer generated, human players, or combination

Real-world Simulations

Attempt to emulate real world operating conditions with great detail.

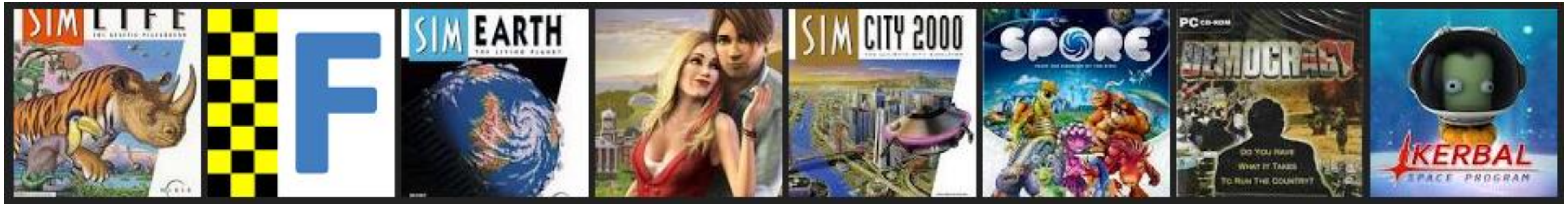
- The more serious the simulation, the more important accuracy is
- Effort may be required to learn all of the intricacies of the game

Most simulate complex machinery, such as cars, aircraft, tanks, etc.

- The result is racing games, flight simulators, war games, etc.

Not all simulations are so serious

- Such games are referred to as arcade simulations



Sports Games

Players to participate in a sporting event or activity

- Can take player, owner, manager, or coaching roles
- Single match, series, entire season of the team or franchise

Prowess in the sport does not translate to the video game, but that is the point, player’s fantasy!

Must accurately reproduce rules and strategies of the sport

- Arcade versions with relaxed rules or reduced realism can also be entertaining

Fighting Games (Sports)

Players control figures on the screen and use a combination of moves to attack opponents and defend from attacks

- Players expect a set of basic attacks and counters to start, as well as more complex combinations to master over time
- Most fights last only a few minutes, but there may be many rounds in a complete bout

Games are generally viewed from the side

- Newer versions have 3D elements and multiple view angles and camera positions

Puzzle Games

Puzzle games exist purely for the intellectual challenge of problem solving.

The puzzles are an end in themselves and are not integrated into a story, as is the case with adventure games.

Puzzles can be non real-time or real-time.

- There are little or no time constraints in non real-time puzzles.
- Real-time puzzles have some timing elements and contain some action.

Online Games

Can include any genres but their core feature is multiplayer network play

- From 2-4 players, to dozens, hundreds, or possibly thousands of players
- Hence the term MMORPG, Massively Multiplayer Online Role-Playing Game.

Often, communities grow around these games

Online gaming is still relatively immature, with many technical and business difficulties unsolved.

Casual Games

Easy-to-play, short session games with little learning curve

- Includes adaptations of traditional games like chess, hearts, and solitaire.
- Includes television games such as Jeopardy or Wheel of Fortune
- Often made addictive by requiring to play often for rewards

Players generally want to drop into and out of these games quickly.

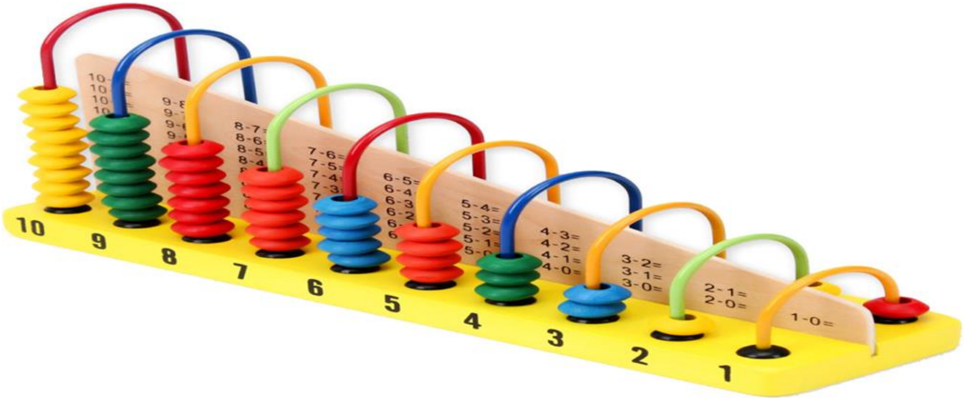
Educational Games

Intended to teach while they entertain at the same time

- Sometimes called edutainment

These games are generally aimed at a younger audience

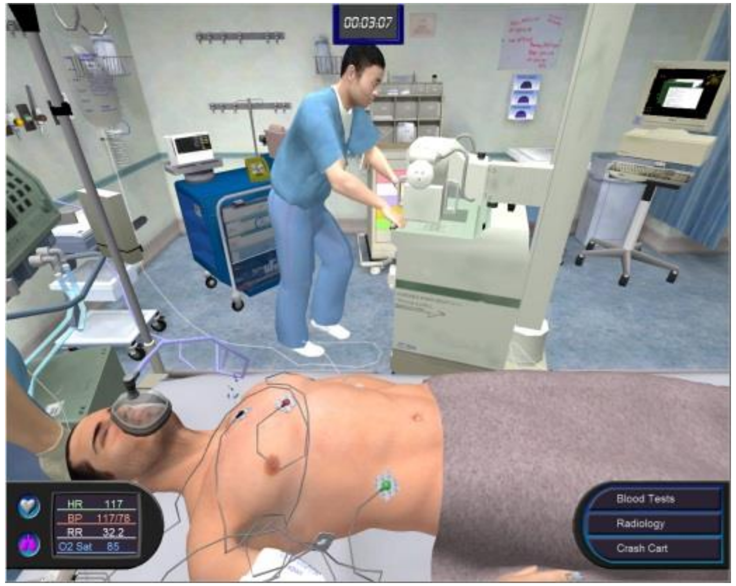
- Designers work closely with target audience to ensure the content is appropriate for the target group



Serious Games

The premise is to apply game design, technologies, and skills to non-entertainment applications. This includes:

- Medical applications.
- Educational applications.
- Social and public policy applications.
- Business and management applications.
- Military applications.
- Plus many other types of simulations and applications.



Combining Genres

https://www.youtube.com/watch?v=WOQwakqWs7k

Programming Basics

Programming Basics Overview

Variables
- Types
- Member vs Local Variables

Classes vs. Objects
- Class diagrams, Subclasses
- Instances

Methods
- Constructors
- Member Methods
- Parameters
- Return Value

Algorithms

The faster you understand these concepts, the faster you can make games!

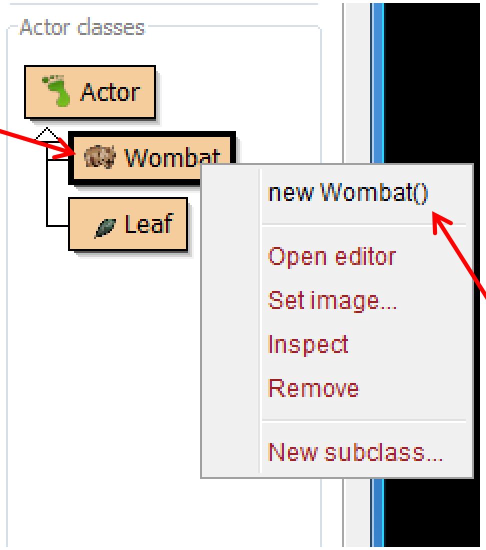
You may not master these concepts at the end of the class, but you must understand them within a few weeks.

These concepts apply to most Programming Languages (including Stride and Java)

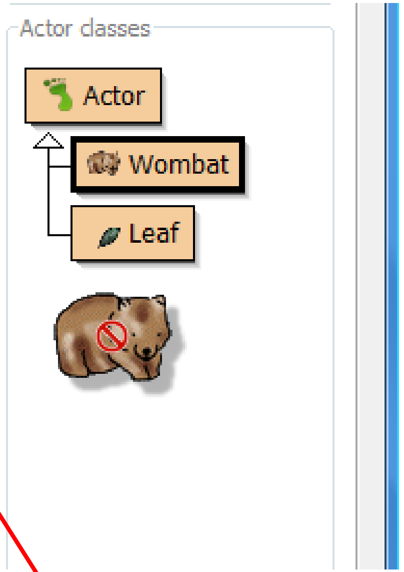
In this lecture, we use Greenfoot and Stride to illustrate the concepts

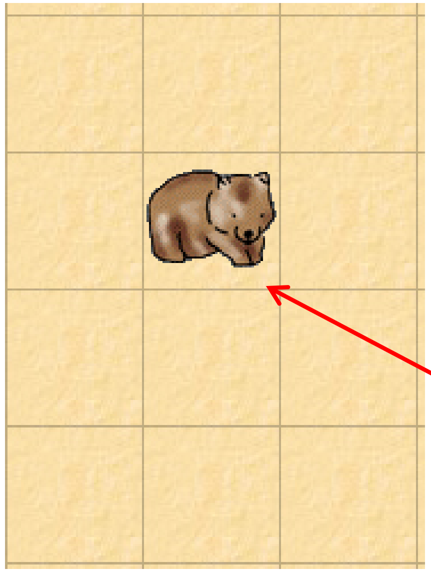
Classes vs. Objects

Right Click on Wombat



Click New Wombat()

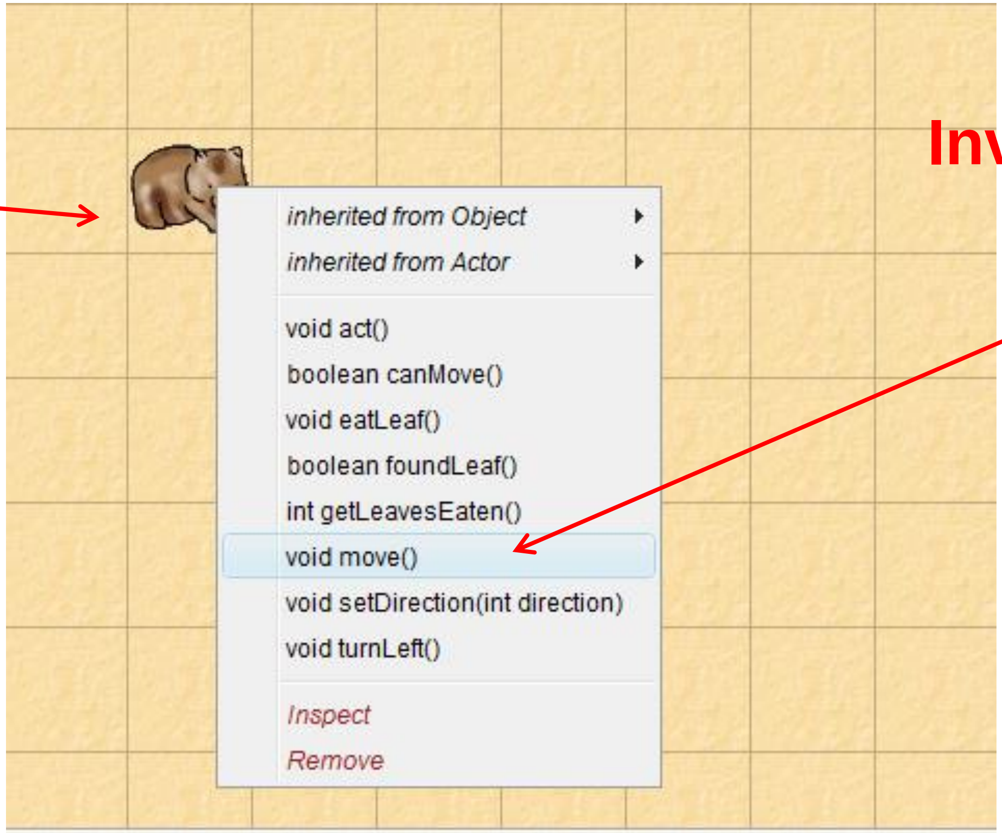




Drag to World

Interacting with Actor Objects (calling methods)

Right Click on the Wombat

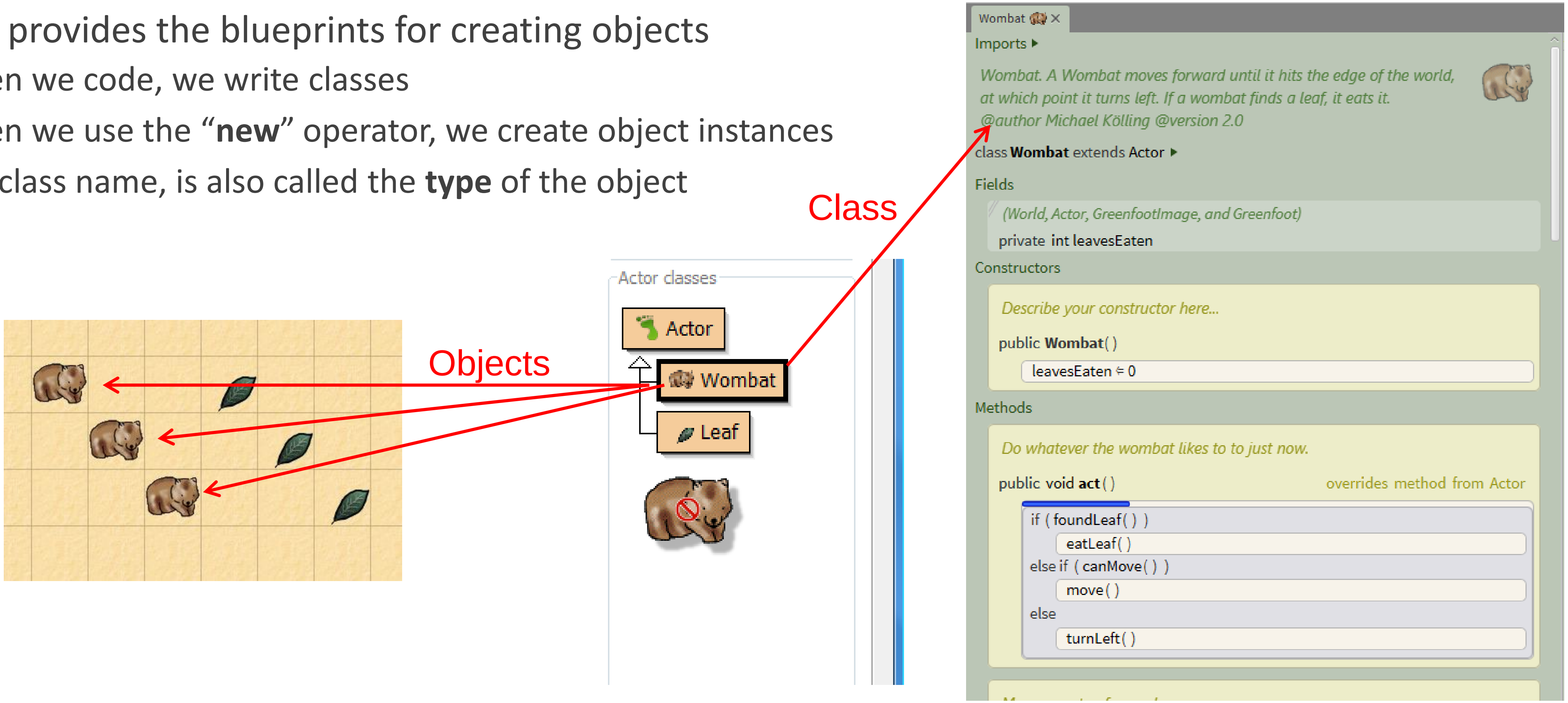


Invoke the Move method

Classes and objects

A class provides the blueprints for creating objects

- When we code, we write classes
- When we use the “**new**” operator, we create object instances
- The class name, is also called the **type** of the object



```
Imports
Wombat. A Wombat moves forward until it hits the edge of the world,
at which point it turns left. If a wombat finds a leaf, it eats it.
@author Michael Kölling @version 2.0
class Wombat extends Actor
Fields
  (World, Actor, GreenfootImage, and Greenfoot)
  private int leavesEaten
Constructors
  Describe your constructor here...
  public Wombat( )
    leavesEaten ⇐ 0
Methods
  Do whatever the wombat likes to to just now.
  public void act( )                    overrides method from Actor
    if ( foundLeaf( ) )
      eatLeaf( )
    else if ( canMove( ) )
      move( )
    else
      turnLeft( )
```

Variables

Variables are used to store information, they have a type and a name.

Types

boolean	: True or false, or from boolean expression (7 < 3)
int	: Integer value (Whole value, no decimals)
double	: Numeric value (including decimals)
String	: Some Text
Actor	: Greenfoot Game Object
Wombat	: User-defined variable, specific to the Wombat game

Local variable : Variable declared within a method, visible locally

Member variables (aka fields) : Variable belonging to an object

Classes and objects
Local variables vs. Member variables (fields)

Local variables

- Methods may contain variables internally, such as parameters and variables declared within the method.

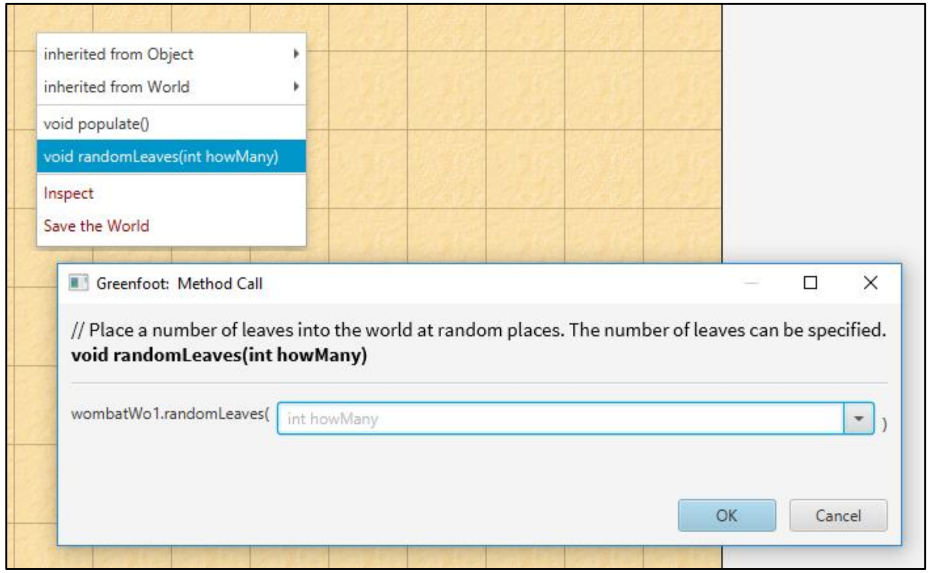
Member variables, or fields

- A class will contain **member variables**, or **fields**, used to define the state of an object.
- **Fields values** for different **objects** are generally independent

Classes and objects – Member Methods

A class will implement **member methods**, that will define behaviors for objects

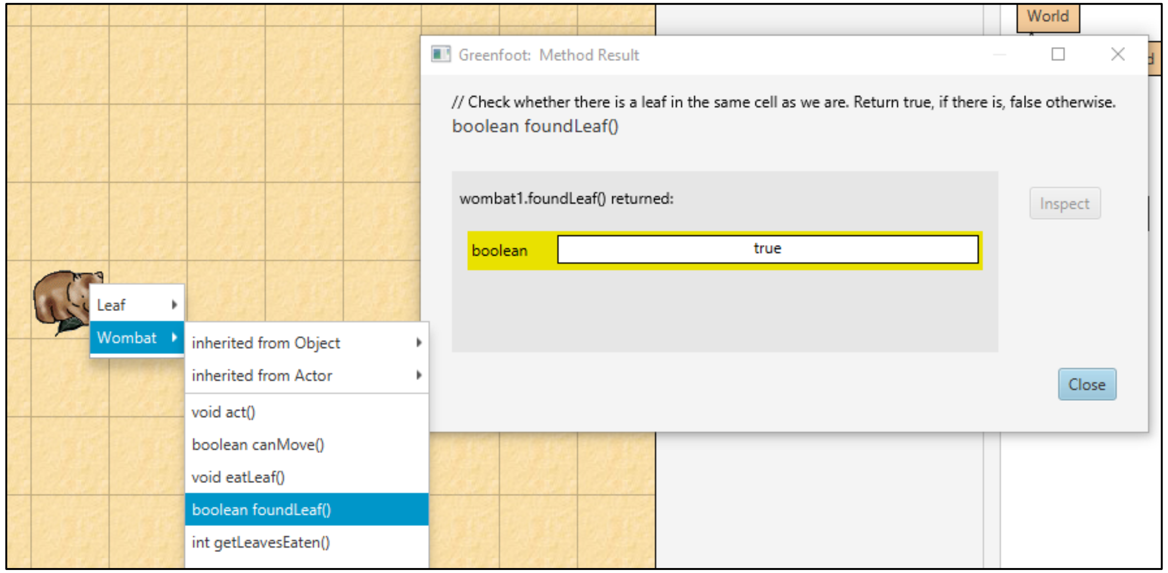
- Methods may require one or many parameters (example on the left)
- Methods may **return** a value, which must be consistent with the **return type** (example on the right)
  - If method doesn't return a value, its return type is **void**



// Place a number of leaves into the world at random places. The number of leaves can be specified.
void randomLeaves(int howMany)

wombatWo1.randomLeaves(int howMany)

Parameter required for randomLeaves method



// Check whether there is a leaf in the same cell as we are. Return true, if there is, false otherwise.
boolean foundLeaf()

wombat1.foundLeaf() returned:

boolean true

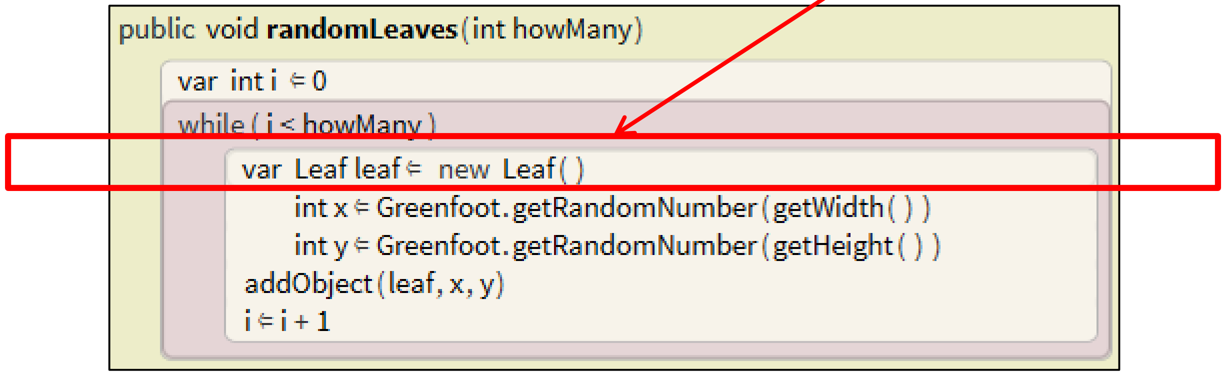
Method foundLeaf returns a boolean value (true or false)

Classes and objects - Constructors

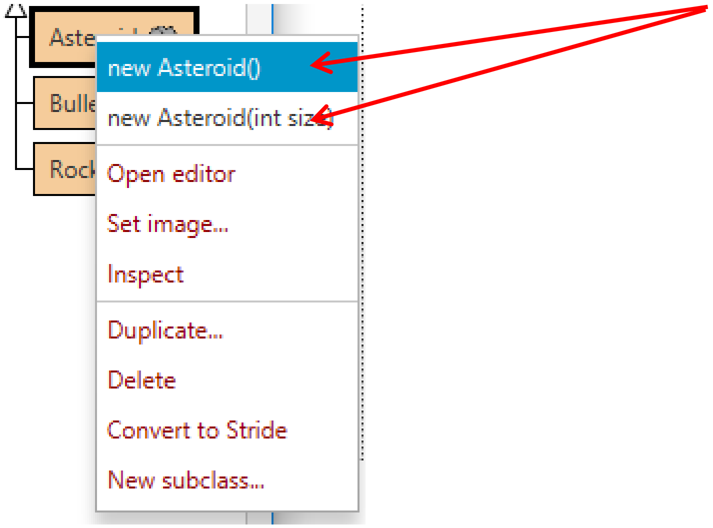
A class will implement a **constructor**, or many constructors, which are methods used to initialize instance objects member variables (fields)

The constructor method doesn't return anything, and has the same name as the class

To instantiate an object, we must call the **new** operator with the name of the class.



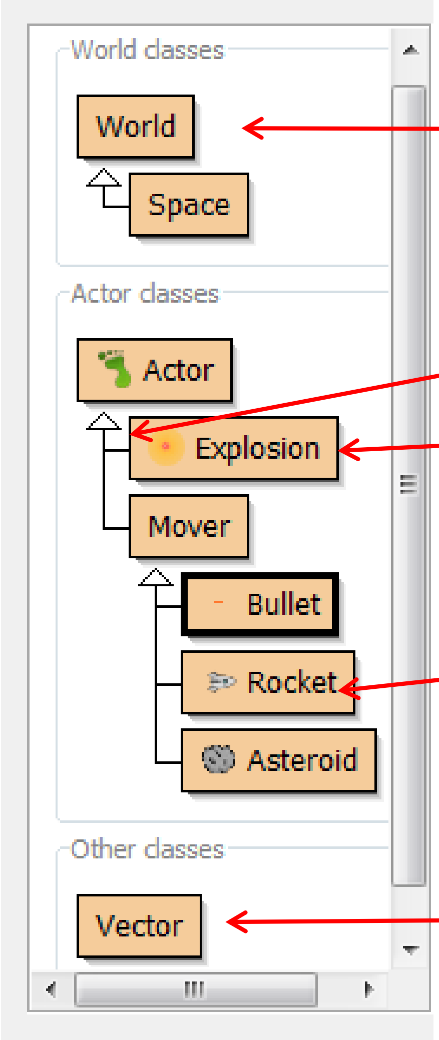
```
public void randomLeaves(int howMany)
    var int i ⇐ 0
    while ( i < howMany )
        var Leaf leaf ⇐ new Leaf( )
            int x ⇐ Greenfoot.getRandomNumber(getWidth( ) )
            int y ⇐ Greenfoot.getRandomNumber(getHeight( ) )
        addObject(leaf, x, y)
        i ⇐ i + 1
```



2 Constructors

Asteroid implements two constructors. One constructor initializes the size based on the parameter value

Understanding the Class Diagram



World Class is always there in Greenfoot scenarios, it is built-in. Space represents a specific world for this scenario

Arrows show relationships

Explosion and Mover are subclasses of Actor

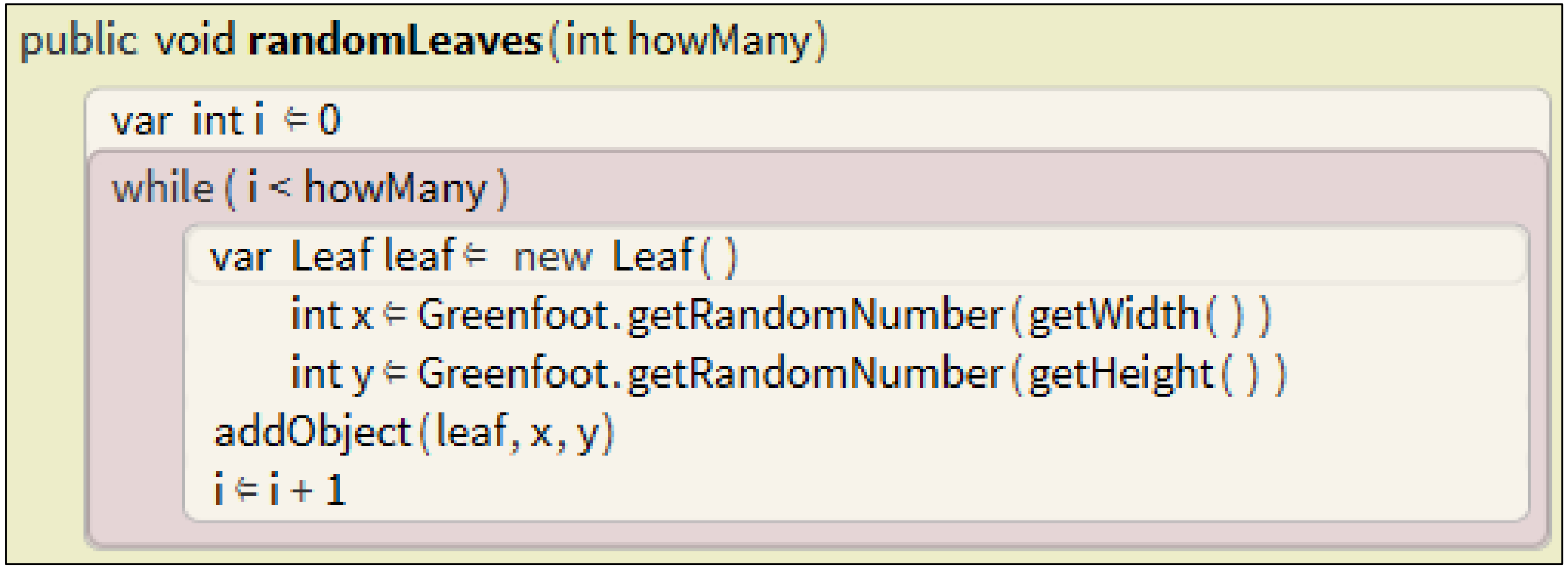
Bullet, Rocket, and Asteroid are subclasses of Mover.

Vector is a helper class

Algorithms

An algorithm is a sequence of instructions (or code statements) that will accomplish a specific task.

For example, the method **void randomLeaves(int howMany)** will add the number of leaves specified as parameter at random locations in the world.



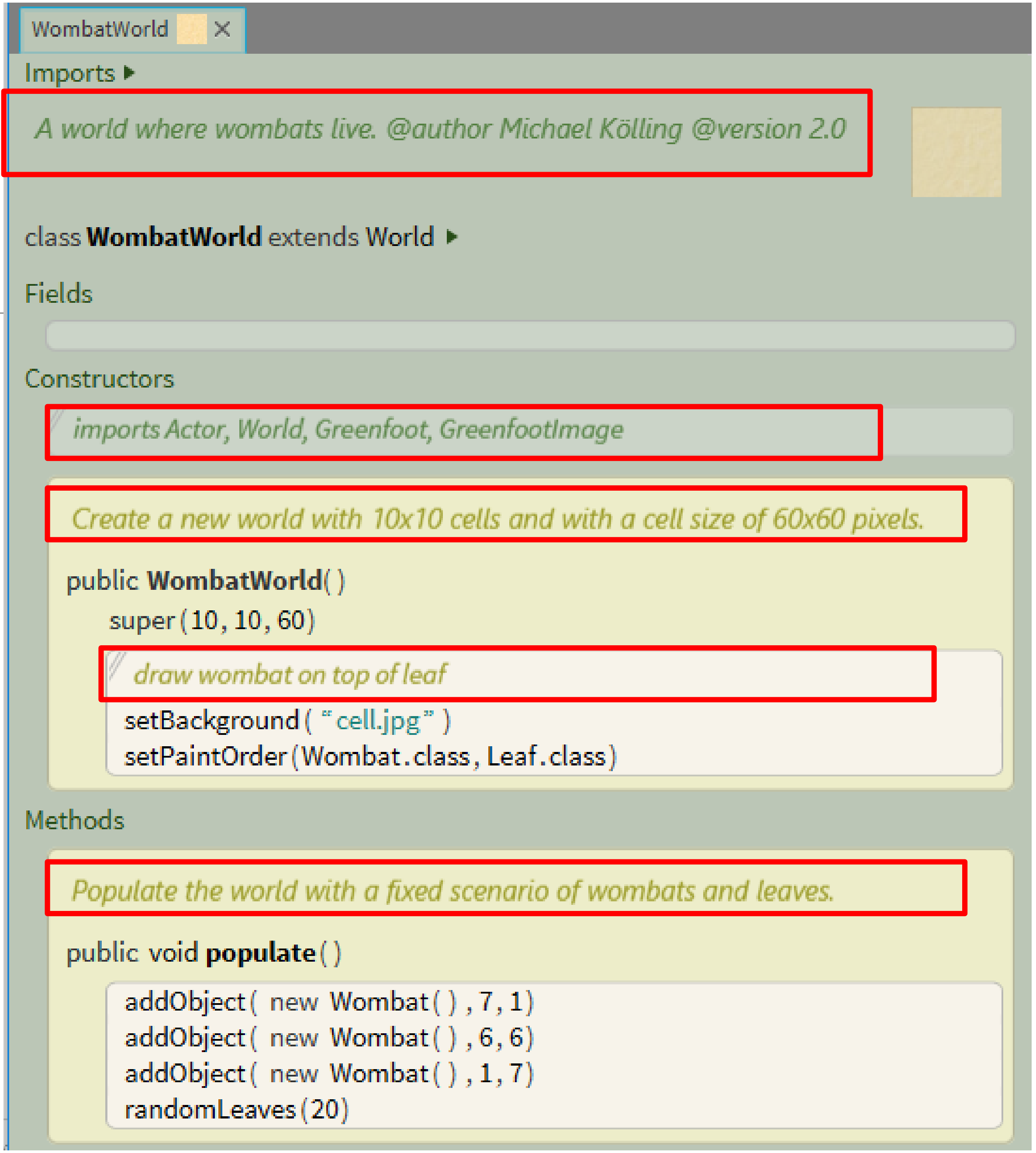
```
public void randomLeaves(int howMany)
    var int i ⇐ 0
    while ( i < howMany )
        var Leaf leaf ⇐ new Leaf( )
            int x ⇐ Greenfoot.getRandomNumber(getWidth( ) )
            int y ⇐ Greenfoot.getRandomNumber(getHeight( ) )
        addObject(leaf, x, y)
        i ⇐ i + 1
```

Comments

Reading code is sometimes difficult and cryptic, comments help clarify!

Comments can be inserted in a block of code to clarify it

Comments are also used to provide (or generate) documentation in a project



```
Imports ▸
A world where wombats live. @author Michael Kölling @version 2.0

class WombatWorld extends World ▸
Fields

Constructors
  imports Actor, World, Greenfoot, GreenfootImage

  Create a new world with 10x10 cells and with a cell size of 60x60 pixels.

  public WombatWorld( )
      super(10, 10, 60)
        draw wombat on top of leaf
        setBackground( "cell.jpg" )
        setPaintOrder(Wombat.class, Leaf.class)
Methods
  Populate the world with a fixed scenario of wombats and leaves.

  public void populate( )
        addObject( new Wombat( ) , 7, 1)
        addObject( new Wombat( ) , 6, 6)
        addObject( new Wombat( ) , 1, 7)
        randomLeaves(20)
```

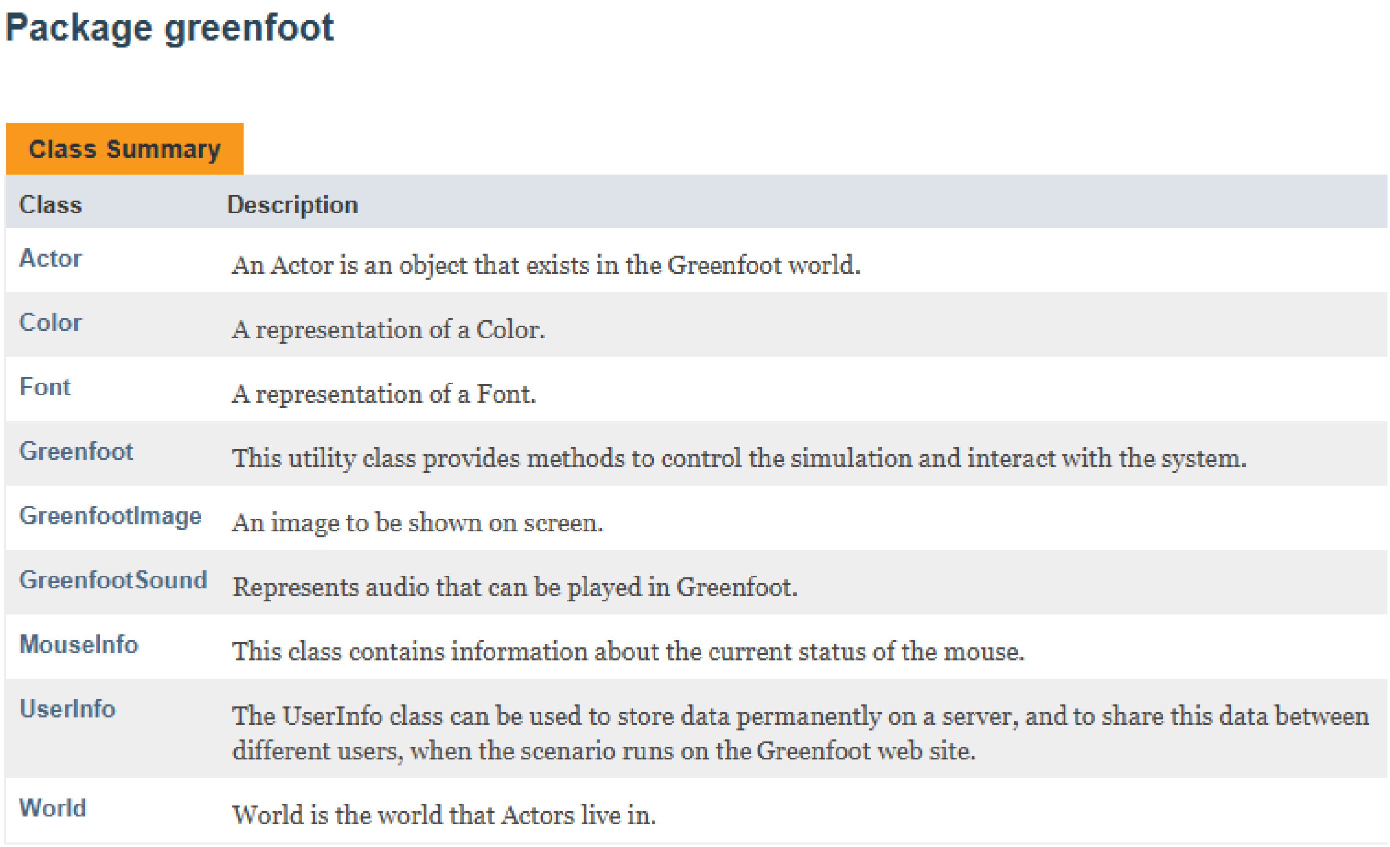
Java Packages – Greenfoot Package

Java includes vast amount of reusable code and features organized as packages.

Anyone can write packages. Some packages are built-in Java, while others are external.

Greenfoot provides classes for making games in the external Greenfoot package.

Right click on the World class, or Actor class to access the greenfoot package documentation

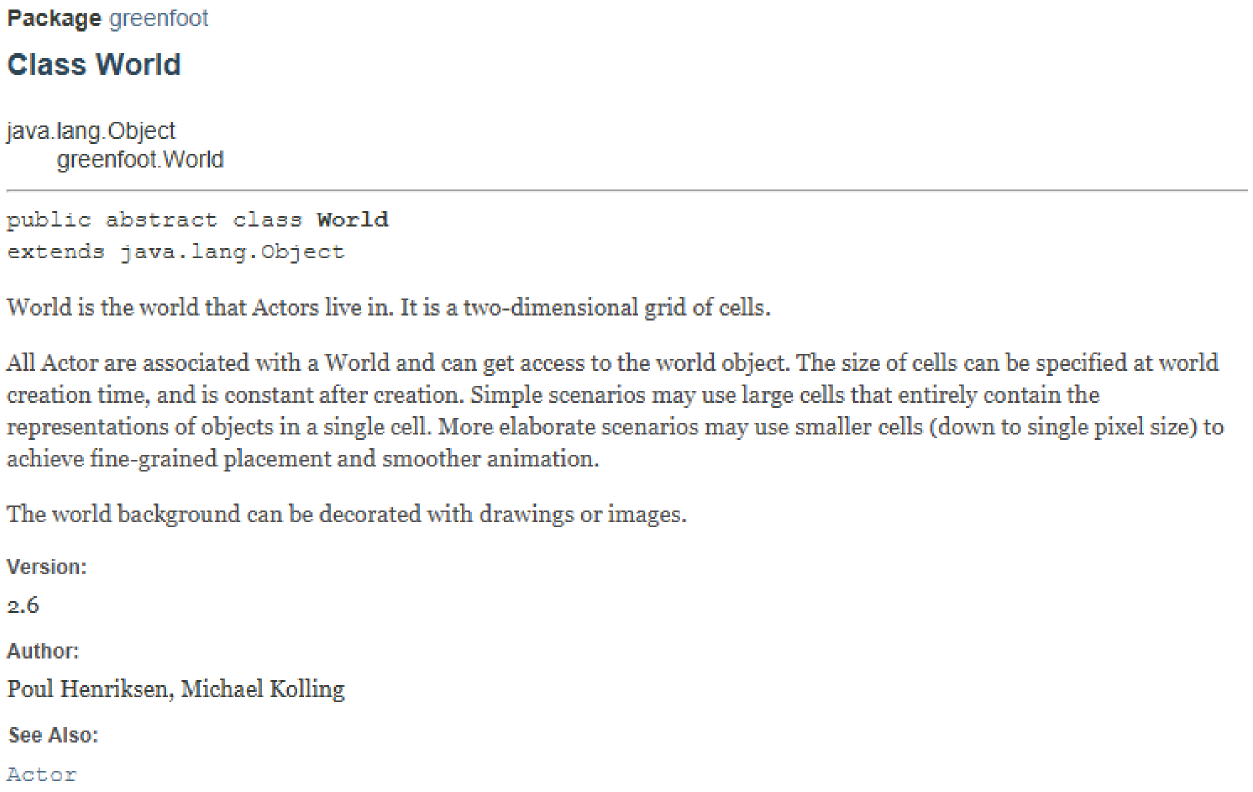


Package greenfoot

Class Summary

Class	Description
Actor	An Actor is an object that exists in the Greenfoot world.
Color	A representation of a Color.
Font	A representation of a Font.
Greenfoot	This utility class provides methods to control the simulation and interact with the system.
GreenfootImage	An image to be shown on screen.
GreenfootSound	Represents audio that can be played in Greenfoot.
MouseInfo	This class contains information about the current status of the mouse.
UserInfo	The UserInfo class can be used to store data permanently on a server, and to share this data between different users, when the scenario runs on the Greenfoot web site.
World	World is the world that Actors live in.

Class Documentation



Package greenfoot

Class World

java.lang.Object
    greenfoot.World

```
public abstract class World
extends java.lang.Object
```

World is the world that Actors live in. It is a two-dimensional grid of cells.

All Actor are associated with a World and can get access to the world object. The size of cells can be specified at world creation time, and is constant after creation. Simple scenarios may use large cells that entirely contain the representations of objects in a single cell. More elaborate scenarios may use smaller cells (down to single pixel size) to achieve fine-grained placement and smoother animation.

The world background can be decorated with drawings or images.

Version:

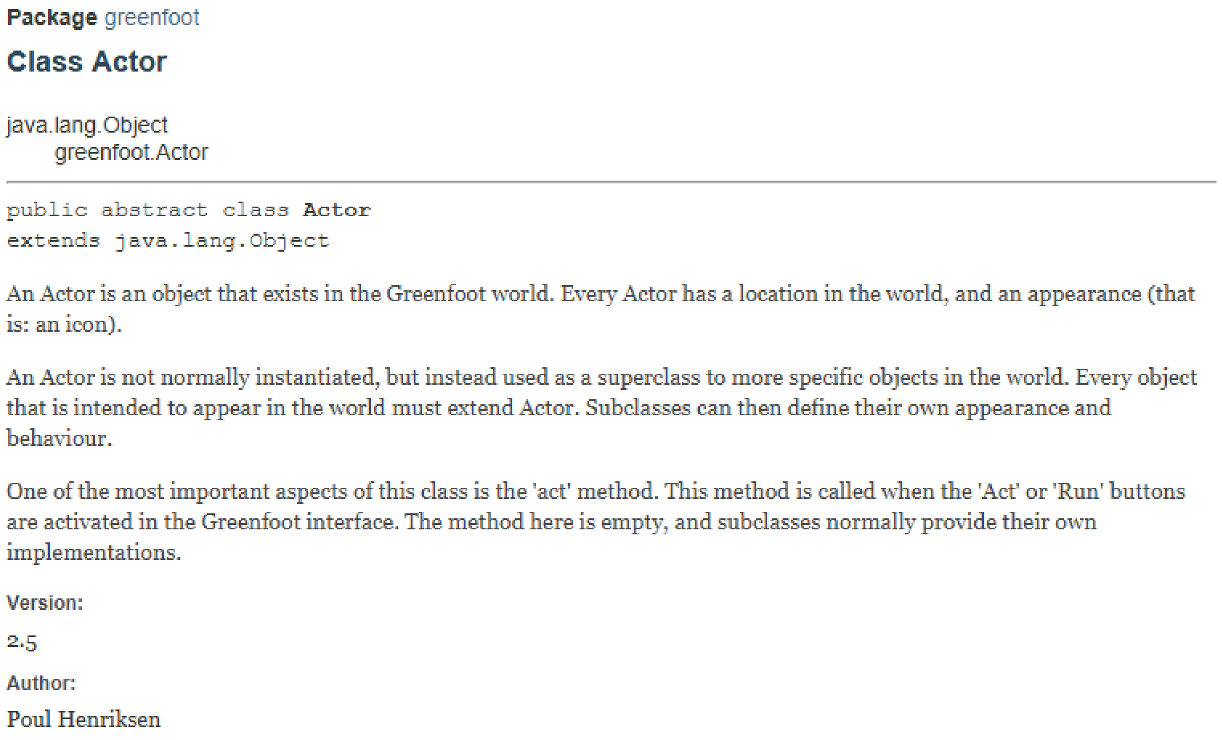
2.6

Author:

Poul Henriksen, Michael Kolling

See Also:

Actor



Package greenfoot

Class Actor

java.lang.Object
    greenfoot.Actor

```
public abstract class Actor
extends java.lang.Object
```

An Actor is an object that exists in the Greenfoot world. Every Actor has a location in the world, and an appearance (that is: an icon).

An Actor is not normally instantiated, but instead used as a superclass to more specific objects in the world. Every object that is intended to appear in the world must extend Actor. Subclasses can then define their own appearance and behaviour.

One of the most important aspects of this class is the 'act' method. This method is called when the 'Act' or 'Run' buttons are activated in the Greenfoot interface. The method here is empty, and subclasses normally provide their own implementations.

Version:

2.5

Author:

Poul Henriksen

Advice: Do not memorize all classes and methods, work with documentation!

Questions

?