Javan GUI Scratchaajalle

Kertausta Javan perusteista

Java-luokan muotoilu

yksi class = yksi hahmo = yksi tiedosto

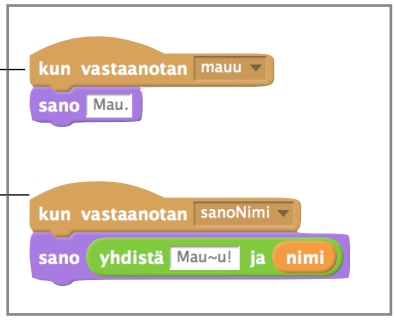
```
public class Kissa {
    private String nimi;

    public Kissa(String annettuNimi) {
        this.nimi = annettuNimi;
    }

    public void mauu() {
        System.out.println("Mau.");
    }

    public void sanoNimi() {
        System.out.println("Mau~u! = " + this.nimi);
    }
}
```

hahmon teko-ohje



lista määritettyjä komentoja = metodeja


```
public class Ohjelma {
    public static void main(String[] args) {
        Kissa aatu = new Kissa("Aatu");
    }
}
```

ohjelmassa saa olla
vain yksi main = vihreä lippu

Muuttujien teko

`int`	kokonaisluku
`double`	desimaaliluku
`String`	teksti
`boolean`	totuusarvo (totta / epätotta)

```
int luku = 3;
String nimi = "Aatu";
```

<-> millainen mikä = mitä

Scratchin ei tarvinnut tietää mitä
muuttujaan laitettiin, Javan tarvitsee.

Jos muuttuja pitää muistaa komennon/metodin ulkopuolella, pitää se määrittää myös esim. henkilökohtaiseksi.

```
private int suunta = 90;
```

Olioiden teko

Scratchin **uusi** (=new) klooni

```
Kissa aatukissa = new Kissa("Aatu");
```

Myös oliot ovat public/private -määriteltyjä, kun ne ovat komennon/metodin ulkopuolella. Esimerkiksi:

```
public class Tamagotchi {
    private Kissa herttuatar;

    public Tamagotchi() {
        this.herttuatar = new Kissa("Herttuatar");
    }
}
```

Olio on luokan (class) perusteella tehty klooni.

Scratch-termeillä:
hahmo = luokka
klooni = olio

Tehtäviä

Tehdään oravapeli. Pelissä orava keräilee pähkinöitä. Osan niistä se voi istuttaa ja toiset säilöä. Istutetut tuottavat uusia pähkinöitä, mutta ne saattaa myös syödä joku toinen orava. Oravan tarvitsee myös itse syödä yksi pähkinä päivittäin.

Tehtävä: Orava_1

Tee  uusi projekti, joka on Java Application. Project Name voi olla Oravapeli. Anna projektin luoda myös Main Class sen nimisenä kuin mitä Netbeans ehdottaa.

Tee projektiin tämän jälkeen uusi luokka, jonka *class name* on "Orava".

Tehtävä: Orava_2

Ohjelmoi **Orava**-luokalle sen henkilökohtainen muisti ja rakennusohje eli konstruktori.

```
public class Orava {
    private ___ _____________________;

    public _____(___ annetutPahkinat) {
        this.______________________ = annetutPahkinat;
    }
}
```

Vertaa tyhjiä kohtia aiempaan kissa-esimerkkiin. Kissalla oli privaatti muisti `nimi`, tee oravalle sen sijaan numeromuuttuja: `pahkinoitaVarastossa`. Täydennä konstruktoriin, että `annetutPahkinat` tallennetaan oravan henkilökohtaiseen muistiin.

Metodi = komento

Scratchissa metodeita ovat:

ja

Metodi voi olla:

`public`	julkinen, kaikille hahmoille
`private`	yksityinen, vain kyseiselle hahmolle

Metodin muotoilu

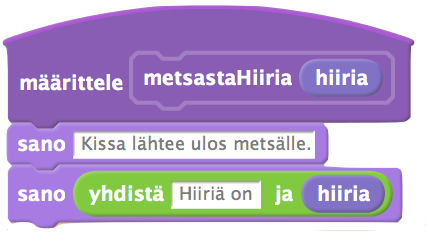
Metodin vastauksena saa tällaisen arvon.

Metodin nimi.

Taustatiedot metodin käyttöön.

```
public class Kissa {
    public int metsastaHiiria(int hiiria){
        System.out.println("Kissa lähtee ulos metsälle.");
        System.out.println("Hiiriä on " + hiiria + ".");
        return hiiria/2;
    }
}
```

Vastauksen palautus. Jos vastaus on void, return-komentoa ei tarvitse olla tai se on return;



Scratchissa ei ole metodia, joka voisi palauttaa (=return) vastausta.

> **Tehtävä: Orava_3**
>
> Tee **Oravalle** komento `public void kerroTilanne()`, joka toimii seuraavasti:
>
> 
>

Metodin/komennon käyttö

Metodia voi käyttää vaikkapa näin toisen komennon aikana:


```
public static void main(String[] args) {
    Kissa aatu = new Kissa("Aatu");
    aatu.sanoNimi();

    Scanner kyselija = new Scanner(System.in);
    System.out.println("Paljonko hiiriä on?");
    int hiiria = Integer.parseInt( kyselija.nextLine() );
    int napattu = aatu.metsastaHiiria(hiiria);
    System.out.println("Hiiriä napattiin: " + napattu);
}
```

Huom! Jos metodi ei ole samassa luokassa kuin mistä sitä kutsutaan, metodille pitää antaa myös tekijä. Esimerkiksi äskeisessä ohjelmassa komennettiin aatua: `aatu.sanoNimi();`

> **Tehtävä: Orava_4**
>
> Kirjoita **Oravapeli**-luokkaan main-metodiin (eli vihreän lipun alle) ensin komento luoda Orava-klooni esim. neljällä pähkinällä: `new Orava(4);` ja anna kloonille sitten komento kertoa tilanne. (Ks. miten äsken kutsuttiin metodia `sanoNimi()`.)
>
> Huomaa, että oravalle ei voi antaa komentoa, jos ohjelma ei uuden oravan luonnin jälkeen muista minne se sen sijoitti. Tallenna siis uusi orava muuttujaan, kuten äsken materiaalissa tallennettiin uusi kissa muuttujaan `aatu`.
>
> Testaa, että peli käynnistyessään käskee oravaa kertomaan tilanteensa ja orava vastaa.

Jossittelut

```
if (syli == true) {
    System.out.println("Hrrrr.");
}

if (ruokaa < 10) {
    System.out.println("Nau?");
}
else if (ruokaa < 20) {
    System.out.println("Hrrrr.");
}
else {
    System.out.println("Kurrrrr. Kurrrrr.");
}
```

Javassa **väliin** voi laittaa myös `else if` -vaihtoehtoja, jos jokin aiemmista väitteistä ei vaikka ole vielä ollut totta.

==	<, <=	>, >=	!=	&&	||	!
onko samat	pienempi	suurempi	onko eri	ja	tai	ei

esim. if (!syli == true) {…

Sisennykset

{ } -sulut kertovat esimerkiksi:
- mikä kaikki kuuluu hahmon koodeihin
- mikä kaikki kuuluu komentoon
- mikä kaikki kuuluu jos- tai ikuisesti-komennon sisään

Vertaa sisennyksiä ja { } -sulkuja Scratchiin:

```
while (true) {
    if (pisteet == 0) {
        pisteet = 10;
    }
    else {
        pisteet = pisteet + 1;
    }
}
```

Java-sisennys

Tehtävä: Orava_5

Kirjoita oravalle (Orava-luokka eli sen tiedostoon) uusi `public` -komento.

Tehtävä: Orava_6

Tarvitset uuden `pahkinoitaMaassa` -muuttujan, jotta orava voi hoitaa puutarhaansa. Muuttuja voi olla `private`, jolloin muut oravat eivät voi suoraan käydä varkaissa. Kirjoita tämä muuttuja sen rivin jälkeen, jolla määrittelit `private pahkinoitaVarastossa` -muuttujan. Määritä, että pähkinöitä on maassa aluksi 0 (kpl).

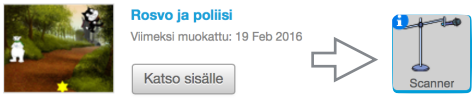
Tee oravalle seuraavat metodit:

```
public void istutaPahkinoita (int maara) {
    jos [maara] > [pahkinoitaVarastossa] , niin
        sano [*sqviik!* Pähkinät tippui lattialle. Joku muu päivä sitten.]
    muuten
        aseta [pahkinoitaVarastossa] arvoon [pahkinoitaVarastossa] - [maara]
        aseta [pahkinoitaMaassa] arvoon [maara]
}

public void poimiPahkinat() {
    sano [yhdistä [Poimitaan pähkinöitä] ja [2] * [pahkinoitaMaassa]]
    aseta [pahkinoitaVarastossa] arvoon [pahkinoitaVarastossa] + [2] * [pahkinoitaMaassa]
    aseta [pahkinoitaMaassa] arvoon [0]
}
```

import Scanner

Jos Scratch osaisi importata, tällainen komento toisi java-käyttäjän util-pelistä Scanner-hahmon käytettäväksi omaan peliisi.




```
import java.util.Scanner;

public class Kissanhoitaja {
    private Scanner lukulaite = new Scanner(System.in);

    public Kissanhoitaja() {
    }

    public void tulkitseKissaa() {
        System.out.println("Sano kissa jotain.");
        String vastaus = lukulaite.nextLine();
        System.out.println("Kyllä se voi ihan: " + vastaus);
    }

    public int montakoKissaa() {
        System.out.println("Montako kissaa?");
        int maara = Integer.parseInt( lukulaite.nextLine() );
        return maara;
    }
}
```

Tällaista tyhjää hahmon teko-ohjetta ei oikeastaan tarvitse kirjoittaa.

Kannattaa esittää ennen skannausta kysymys, että pelaaja älyää vastata.

`Scanner`	skanneri, lukulaite
`nextLine`	(lue) seuraava rivi
`Integer`	kokonaisluku
`parseInt`	jäsennä kokonaisluku, parsi kasaan luku

Toistot = silmukat

```
for (int i=1; i<=10; i++) {
    System.out.println("Boink!");
}
```

i on vähän niin kuin hyppynarun sisällä pomppiva ihminen, joka laskee hyppyjä. Kun hyppyjä on tarpeeksi, voi lopettaa.

```
while (true) {
    System.out.println("Hrrrr.");
}
```

while:n ( ) -sulkujen sisälle voisi laittaa myös testin, silloin ohjelma sanoisi `Hrrrr.` kunnes testi ei olisi enää totta.

Toistojen poikkeuksellinen käsittely:

`break;`	pysäyttää toiston
`continue;`	jatkaa toiston seuraavalle kierrokselle

Tehtävä: Orava_7

Pelissä ei vielä käytetä äsken tehtyjä metodeja. Luo main -metodissa Oravan luonnin jälkeen myös Scanner -olio.

Kirjoita sitten Oravapeli-luokkaan seuraavanlainen oravan päivän työjärjestys. Muista, että metodi-komennot (lähetä- ja lohkopalikat) pitää antaa oravalle, eli komento pitää kirjoittaa tyyliin: tekijä.komento(); Jos oravasi nimi on esimerkiksi kurre, komento olisi `kurre.poimiPahkinat();`

Kuten näet pelaajalta pitää kysyä myös istutettavien pähkinöiden määrä. Tähän tarvitset ohjeessa äsken esiintynyttä Scannerin lukuja lukevaa komentoa.

Uutta:

Testaa peliä.

Noppa

Noppaa ei tarvitse suunnitella itse vaan se löytyy lainattavasta java-kirjastosta:

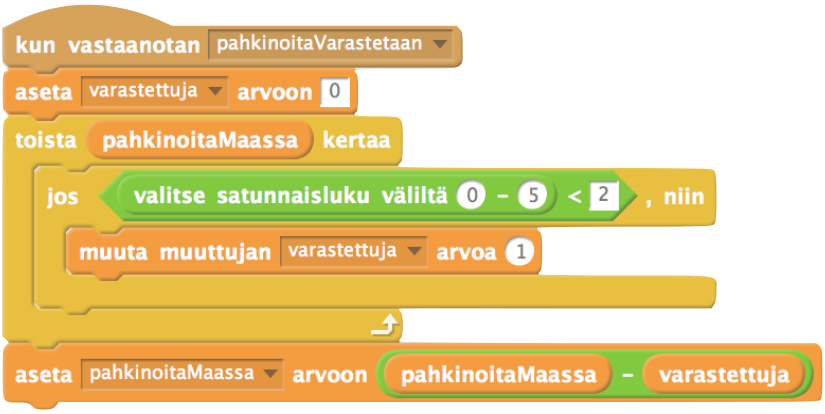
```
import java.util.Random;
```

Java-kirjaston ollessa importattuna (import = tuoda, tuontitavara) nopasta voi tehdä itselleen kopion ja kopiolle voi sitten antaa käskyn antaa satunnainen numero:

```
Random noppa = new Random();
int tulos = noppa.nextInt(6) + 1;
```

Tehtävä: Orava_8

Nyt peliin tarvitaan vielä vähän riskejä. Yön aikana toiset otukset saattavat löytää istutetut pähkinät. **Orava**-luokka on toistaiseksi ollut vastuussa pähkinöistä, joten kirjoitetaan ohje varastapauksiin reagoimisesta sille.



Lisätään myös **Oravapeli**-luokkaan päivän päätteeksi `while(true)` -silmukan loppuun:

Tehtävä: Orava_9

Peli halutaan kenties myös päättää, jos oravan varastot eivät riitä. Lisää syoPahkina-metodiin `System.exit(0)`, jos ruokaa ei ollut jäljellä.

Bonus: Orava_10

Olemme tehneet Orava-luokan (class) eli ohjeen oravan luontiin. Samalla ohjeella voisi tehdä vaikka kokonaisen metsällisen oravia. Tehdään nyt kaksi.

Jäsennellään ensin hieman Oravapeliä, jotta sen pituus ei räjähdä, kun kaikki komennot pitää antaa kahdelle oravalle. Tee Oravapeliin metodi:

```
private static void suoritaPaiva(Orava oravaVuorossa, Scanner kuuntelija) {

}
```

ja siirrä siihen kaikki while-silmukan komennot. Koska metodissa orava tunnetaan nimellä `oravaVuorossa` niin korjaa se myös siirrettyihin metodikutsuihin kuten:
`oravaVuorossa.poimiPahkinat();`

Bonus: Orava_11

Kirjoita while-silmukkaan komento `suoritaPaiva(orava1, kuuntelija);` Korjaa nimien `orava1` ja `kuuntelija` tilalle ne nimet, jotka olet itse antanut ensimmäiselle Orava-oliolle ja Scanner-oliolle.

Testaa, että ohjelma toimii yhä.

Bonus: Orava_12

Luo toinen Orava-olio sen rivin jälkeen kuin millä luotiin ensimmäinen orava. Anna tälle eri nimi.

while-silmukan sisällä (ks. kuva) tulosta aina ensin kumman oravan vuoro on, suorita sitten päivän komennot ja lopuksi tulosta yksi tyhjä rivi. Toista sama toiselle oravalle.

Tyhjät rivit auttavat pelaajia erottamaan rivinsä toisistaan.