Another OO Example

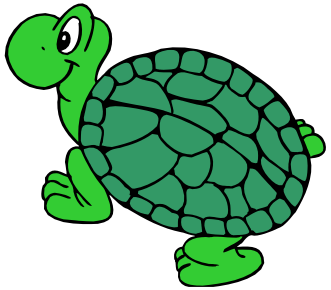
covering

** steps to write OO programs
** overloading ** overriding

Chapter 2 (sections 2.1–2.3; 2.5–2.8; 2.10–2.11) – “Core Java” book
Chapters 2-4 – “Head First Java” book
Chapter 5+10+11 (sections 5.8, 10.11, 11.4-11.5) – “Introduction to Java Programming” book
Chapter 3 – “Java in a Nutshell” book

The Rabbit and the Turtle example

- Sam is a turtle.
- Sam is **green** with a scaly tail and a top speed of 2 miles per hour.
- Peter is a rabbit. Peter is **grey**, has rough fur, a fluffy tail and can run at a speed of 150 miles per hour.
- Both rabbits and turtles can swim, but only rabbits sleep.



Objects?
Attributes?
Operations?

Class diagrams

For the Rabbit?

Rabbit
String name; **String tailType;** **Color color;** **int speed;** **String furType;**
run(); **sleep();** **swim();**

For the Turtle?

Turtle

Rabbit: Class Definition and Information Hiding

```
import java.awt.*;
/**
 * Title:        Rabbit.java
 * Description:  This class contains the definition of a rabbit.
 * Copyright:    Copyright (c) 2001
 * @author       Laurissa Tokarchuk
 * @version      1.0
 */
public class Rabbit {
  // Declaration of instance variables
  String name, tailType, furType;
  Color color;
  int   speed;

  // Declaration of methods - blank for now
}
```

Rabbit
String name; String tailType; Color color; int speed; String furType;

run();
sleep();
swim();

```
public class Rabbit {
  private String name, tailType, furType;
  private Color color;
  private int   speed;
}
```

Controlling our creatures …

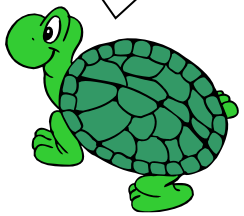
This slide has lots of animation.

Don't tell me how to run!

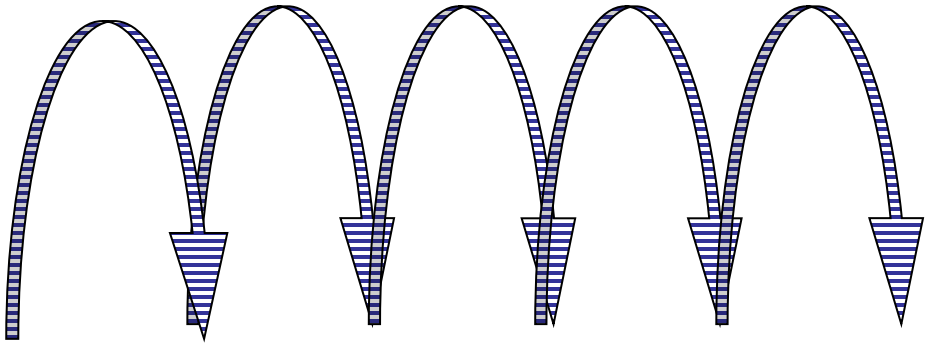
I hop in the air when I run!

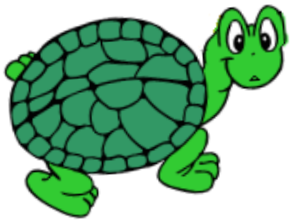
I most certainly do **not** hop!

Or me! I will go at my own pace!



Rabbits and Turtles provide an interface for a client to request that they run. But the client has no control over how they do that!





I am sooo embarrassed!

Without **data hiding**: If the client could control the creature objects, it could make our poor turtle hop! ☹

Queen Mary University of London

Accessor and Mutator methods for rabbit

```
/**
 * This method gets the furType of the rabbit.
 * @return String  Type of fur.
 */
public String getFurType() {
   return furType;
}
```

```
   /**
    * This method sets the furType of the rabbit.
    * @param furtype  Type of fur the rabbit should have.
    */
   public void setFurType(String furType) {
     // check to see that the furType is valid for rabbits
     if ((furType.equals("scaly") || (furType.equals("bald")) {
       System.out.println("ERROR: Illegal fur type.");
     }
     else this.furType = furType;
   }
```

Rabbit Methods

```
/**
  * This is the sleep method for the rabbit. It dictates the
  * number of minutes the rabbit sleeps.
  * @param duration  The number of minutes to sleep.
  */
public void sleep(int duration) {
   // Code of sleep
}
```

```
/**
  * This method allows the rabbit to run. The distance the
  * rabbit runs depends on how long the rabbit runs for, and
  * on whether or not it is running in a zigzag.
  * @param duration  The number of minutes to run.
  * @param zigzag    Whether to run in a zigzag pattern
  * @return int      Number of miles run..
  */
public int run(int duration, boolean  zigzag) {
  // code of run
}
```

Method overloading

- Whenever two or more methods have the same name but different input parameters. For example,

```
public int run(int duration,  boolean  zigzag) { }
public int run(int duration){ }
```

- Both of these methods can exist in the class **Rabbit**.
  - Which one is called depends on how you call it, e.g.

```
Rabbit bugs = new Rabbit();
int distance = bugs.run(5, true);  // OR
int distance2 = bugs.run(5);
```

Rabbit class (in full)

```
import java.awt.*;
public class Rabbit {
  private String name, tailType, furType;
  private Color color;
  private int   speed;

  public String getFurType() { return furType; }

  public void setFurType(String furType) {
    // check to see that the furType is valid for rabbits
    if ((furType.equals("scaley") || (furType.equals("bald")) {
      System.out.println("ERROR: Illegal fur type.");
    }
    else this.furType = furType;
  }
  public void sleep(int duration) {
    // code of sleep
  }
  public int run(int duration,  boolean  zigzag){
    // code of run
  }
  public void swim() {
    // code of swim
  }
}
```

You should always write full comments in the program. Some comments have been removed here to save space on the slide.

Writing our Test class …

```
public class RabbitTest {
  public static void main(String[] args) {
    Rabbit bunny = new Rabbit();
  }
}
```

What are the values of **`name`**, **`furType`** and **`speed`**?

All instance variables are set to their default values, unless otherwise specified.

Initialisation and Constructors

- In Java, all variables must be initialised before they can be used.
- Java automatically sets some initial values for you for variables of the class (instance variables), but not for variables in methods.

field type	initial value
boolean	false
char	'\u0000'
byte, short, int, long	0
float / double	+0.0f / +0.0d
object reference	null

Using Objects

- So by default, our **`Rabbit`** class has a constructor provided by Java. Thus we can create a **`Rabbit`** object as follows:

```
Rabbit  bunny;          // bunny is 'null' now.
bunny = new Rabbit();   // creates a rabbit
```

or equivalently:

```
Rabbit bunny = new Rabbit();
```

A **`Rabbit`** object (or any object) is a reference variable.
This is different from a primitive variable (e.g. **`int`**, **`double`**).

Rabbit's creation

This slide has lots of animation.

```
Rabbit bunnie = new Rabbit();
```

Rabbit

Tells the JVM to allocate space for a reference variable of type **Rabbit** called **bunnie**

Assigns the new **Rabbit** object to the reference variable **bunnie**

Rabbit object

Tells the JVM to allocate space for a new **Rabbit** object on the heap

Now they are joined!

- The variable **bunnie** now controls the **Rabbit** object.

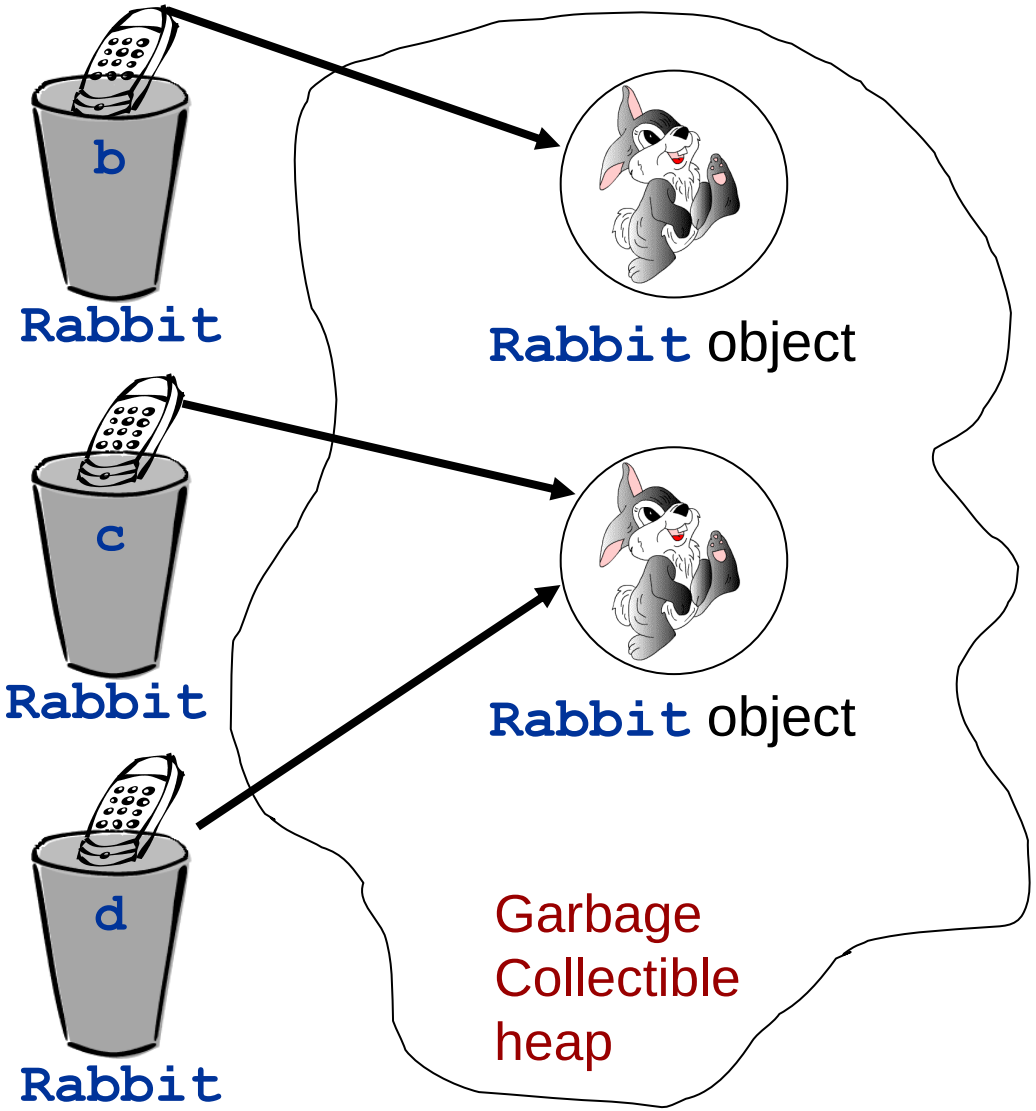
```
bunnie.sleep(5);
```

Rabbit

Life on the Heap! (1/2)

This slide has lots of animation.

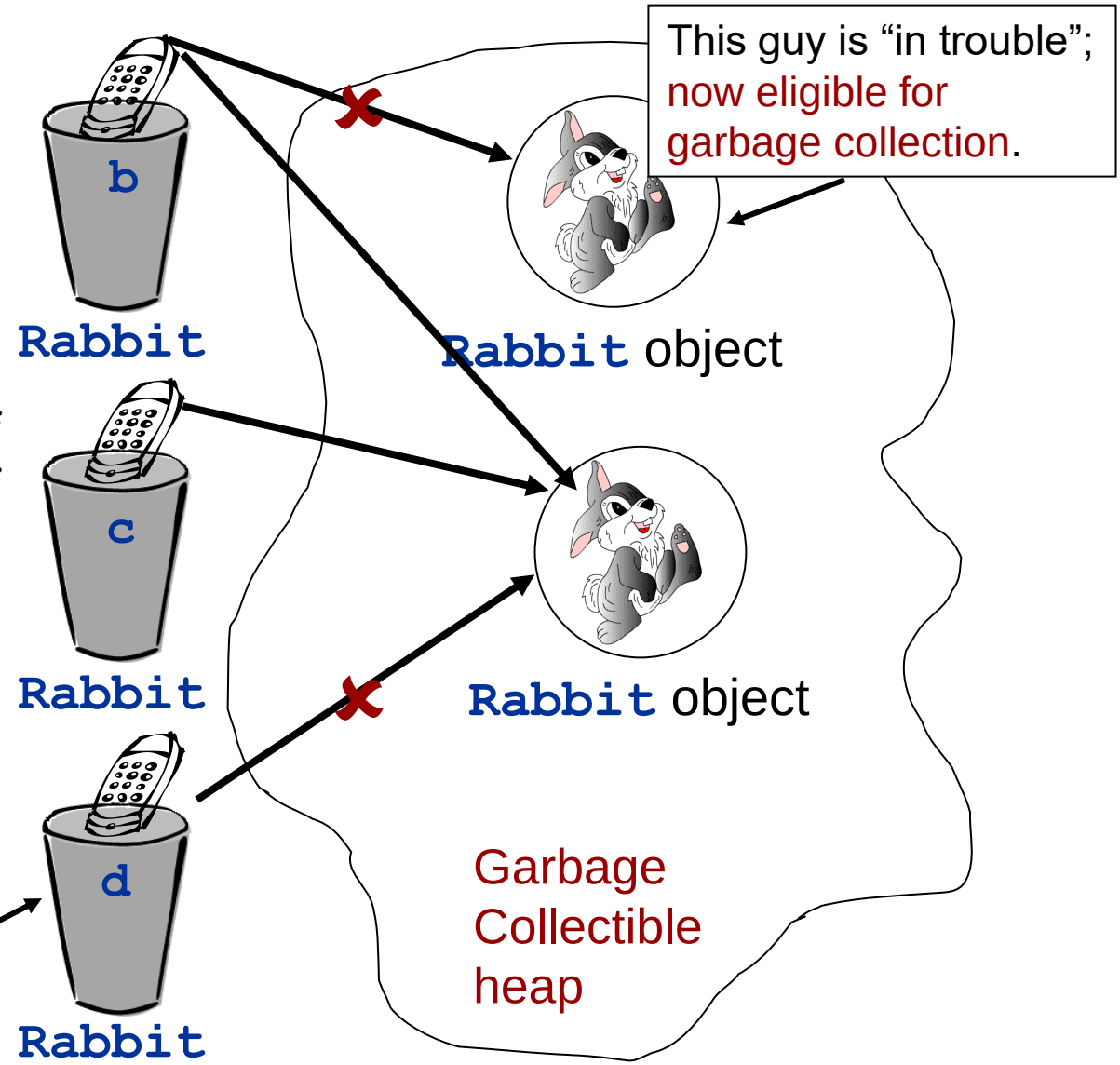
```
Rabbit b = new Rabbit();

Rabbit c = new Rabbit();

// d and c are different
// variables, that refer
// to the SAME object.

Rabbit d = c;
```



Life on the Heap! (2/2)

This slide has lots of animation.

```
Rabbit b = new Rabbit();
Rabbit c = new Rabbit();
// d and c are different
// variables, that refer
// to the SAME object.
Rabbit d = c;
b = c;
d = null;
```



Common Error when accessing methods on objects

Rabbit

```
Rabbit bugs;
bugs.sleep(5);
```

- The **bugs** rabbit cannot sleep!
- In fact, any attempt to use **bugs** will result in:

```
Exception in thread "main"
  java.lang.NullPointerException
```

Don't try to use reference variables before initialising them.

Practice Exercises

- Write the `Turtle` class and test it.

We will only start this in class … students will complete this as homework.

General steps when writing classes (1+2)

- Step 1: Think!
  - States and behaviour of the object
  - States → instance variables
    - How many?
    - Type?
    - Private or public?
  - Behaviour → methods
    - How many?
    - Return type?
    - Parameters?

- Step 2: Skeleton (or basic) code
  - Define a class
  - Declare instance variables
  - A set of constructors
    - How many?
    - Parameters?
    - Type?
  - Write a test program (with a `main()` method) to test it
    - Create new objects using provided constructors

General steps when writing classes (3+4)

- Step 3: Accessors and mutators
  - A set of accessors and mutators
    - How many?
    - Return type?
    - Parameters?
  - Test them in the test program
    - Test each accessor/mutator method
- Step 4: Service methods
  - Write ONE service method first
  - Test it
  - Write another one
  - Test it
  - Etc …
- A method should only do one thing and do it well
  - If a method does too much… consider breaking it down into several smaller methods!

General steps when writing classes (5+6)

- Step 5: `toString()` method
  - Write it, if necessary
- Now you should have a basic OO program working

- Step 6: Improvement
  - Have a full working basic class first
  - Any improvements?
  - Any better solution?
  - Provide user friendly messages?
  - …