Ollscoil na Gaillimhe
University of Galway

CT2106
Object Oriented Programming

Dr. Frank Glavin
Room 404, IT Building
Frank.Glavin@University*of*Galway.ie

School of Computer Science

University *of*Galway.ie

Variables and Types

- A variable is a symbol used to store a value
  - E.g. x = 5

- In **strongly typed** language, you have to tell the compiler/interpreter what **type** the variable is

- The Compiler/Interpreter knows how much space to allocate it in memory

Java Primitive Variables

Type	Size	Range
boolean	1 bit	true or false
byte	8 bits	[-128, 127]
short	16 bits	[-32,768, 32,767]
char	16 bits	['\u0000', '\uffff'] or [0, 65535]
int	32 bits	[-2,147,483,648 to 2,147,483,647]
long	64 bits	$[-2^{63}, 2^{63}-1]$
float	32 bits	32-bit IEEE 754 floating-point
double	64 bits	64-bit IEEE 754 floating-point

Default values

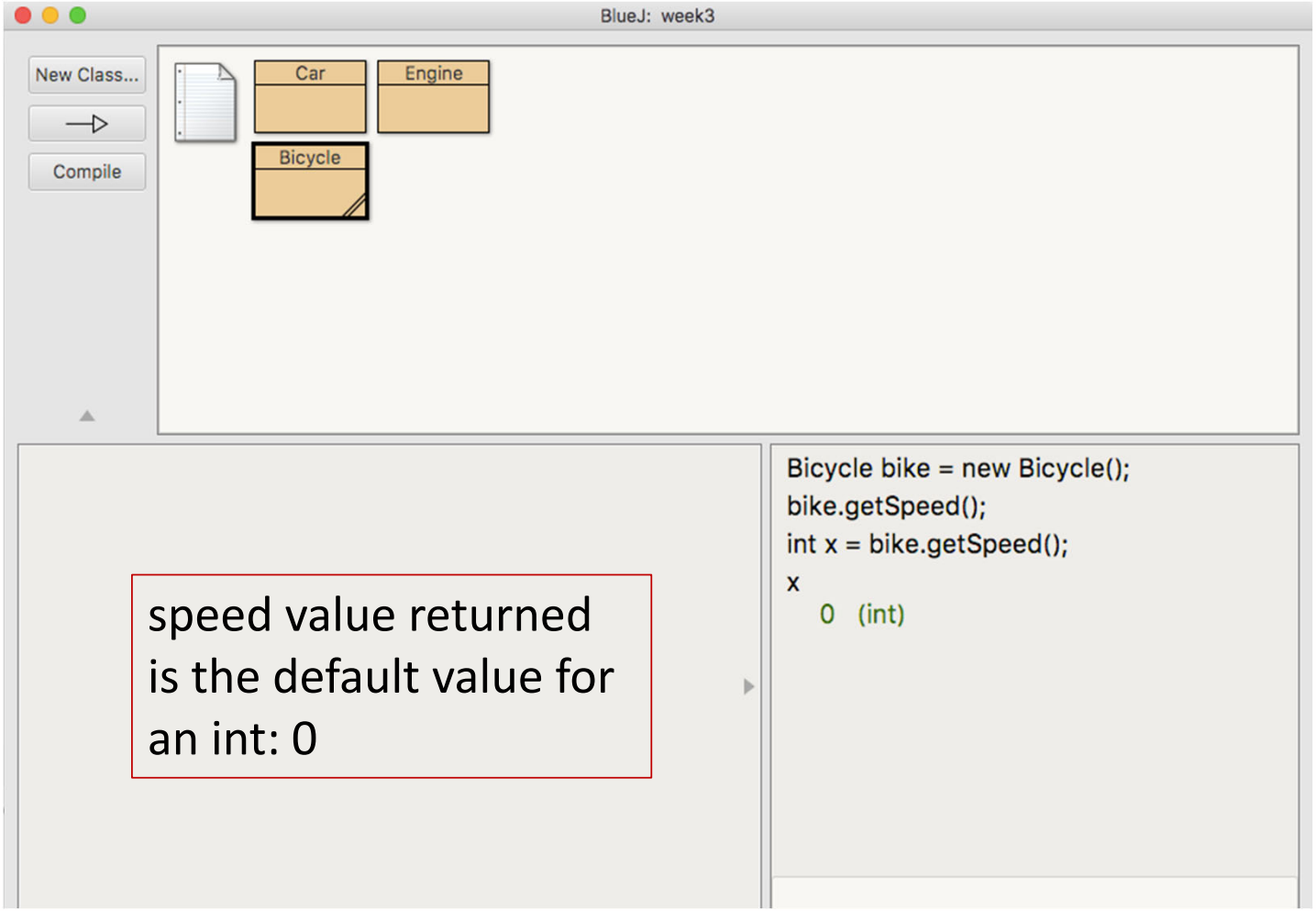
- Each primitive variable has a **default value**.
- The default value is used **only when the variable is used as a field (instance variable)**
- If the field is not explicitly assigned a value, the default value is used
- For example, the default value for an **int** variable is 0 (zero)

Useful example and summary:
https://www.codejava.net/java-core/the-java-language/java-default-initialization-of-instance-variables-and-initialization-blocks

Example

```java
public class Bicycle
{
    // instance variables - replace the example below with your own
    private int speed;

    /**
     * Constructor for objects of class Bicycle
     */
    public Bicycle()
    {
        // note how the speed variable is not initialised
        // it will us the default value for an int,zero

    }

    /**
     * @return  value of speed field
     */
    public int getSpeed()
    {

        return speed;
    }
}
```

Ollscoil na Gaillimhe
University of Galway



speed value returned is the default value for an int: 0

Default Values

- The Code pad in Blue J automatically initialises **variables just as if they were instance variables**.
- This will not happen in a true Java program!
- But it is useful for learning the default values.

Default Values

Your turn – type a variable of each type into Code Pad
E.g type: **int y;**
Hit return
then type: **y**
Hit return
Write down the default value returned for each type

Type	Size
boolean	1 bit
byte	8 bits
short	16 bits
char	16 bits
int	32 bits
long	64 bits
float	32 bits
double	64 bits

Starting Example

```
int y;
    Note: Codepad variables are automatically initialized
     in the same way as instance fields.
y
    0   (int)
```

```
int y;
    Note: Codepad variables are automatically initialized
     in the same way as instance fields.
y
    0   (int)
boolean bool;
bool
    false   (boolean)
byte b;
b
    0   (byte)
short s;
s
    0   (short)
char c;
c
    '\u0000'   (char)
long lg;
lg
    0   (long)
float f;
f
    0.0   (float)
double d;
d
    0.0   (double)
```

Java Primitive Variables

Default values

Type	Size	Range	Default
boolean	1 bit	true or false	false
byte	8 bits	[-128, 127]	0
short	16 bits	[-32,768, 32,767]	0
char	16 bits	['\u0000', '\uffff'] or [0, 65535]	'\u0000'
int	32 bits	[-2,147,483,648 to 2,147,483,647]	0
long	64 bits	$[-2^{63}, 2^{63}-1]$	0
float	32 bits	32-bit IEEE 754 floating-point	0.0
double	64 bits	64-bit IEEE 754 floating-point	0.0

Reference/Object Types

- A reference type is a data type that's based on a class rather than on one of the primitive types that are built into the Java language.
- In fact, there are four categories of reference type:
  - Object Types
  - Interface Types
  - Enum Types
  - Array Types
- For now, we will focus on Object types, the others will follow easily

Object Reference Type: Key points

- A variable that is a reference type is a variable that **points to an object**
- A primitive variable contains **the value** of the primitive type .
- e.g. **int x = 7**; x contains the int value 7
- A reference variable **contains the value of the memory location** of an object
- E.g. **Wheel wheel = new Wheel();**
- The **wheel** variable contains the value of the memory location of the new Wheel object

Key point to Remember

- A reference variable **does <u>not</u> contain the value of the object**
- A reference variable **contains the <u>value of the memory location of the object</u>**
- It is a **pointer**

Null Default value

- The default value of all reference variables is **null**;
- null is a special value in Java
- It means 'No object'
- When you first declare a reference variable, its value is null

```
Bicycle bike; // declaring a reference variable called bike of type Bicycle
bike // what's the value of bike?
    null
Bicycle bike2; // declaring another reference variable of type Bicycle
bike2 // what's the value of bike2
    null
```

NullPointerException

- One of the most common errors generated when running a program in Java is **NullPointerException**
- This error is thrown when your program encounters a reference variable that has not been initialised
- This means that the variable points to its default value = **null**
- Your program then tries to get the object that the variable is pointing to to do something.
- But the object doesn't exist. Variable actually points to null.
- This causes Java to generate a **NullPointerException**

Example

Using your previously defined Bicycle class, type the following into Code Pad

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = null;
bike2 = null;
```

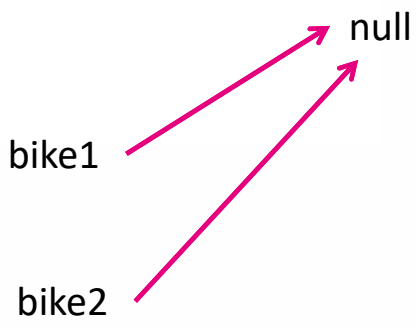
bike1, bike2 are assigned to point to the Bicycle objects just initialised

bike1, bike2 again point to null

What has happened to the previously initialised Bicycle objects?

Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

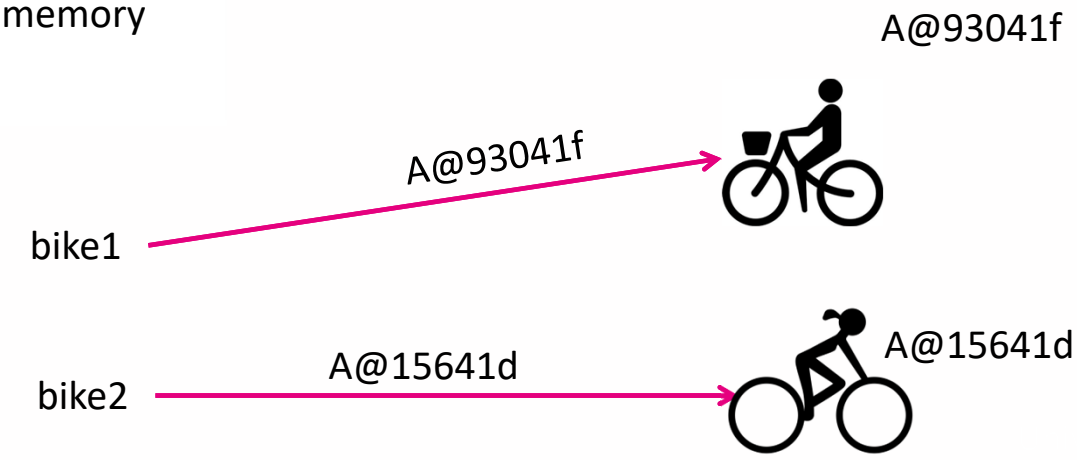


Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

The actual values of bike1, bike2 are memory locations

Objects stored at memory locations



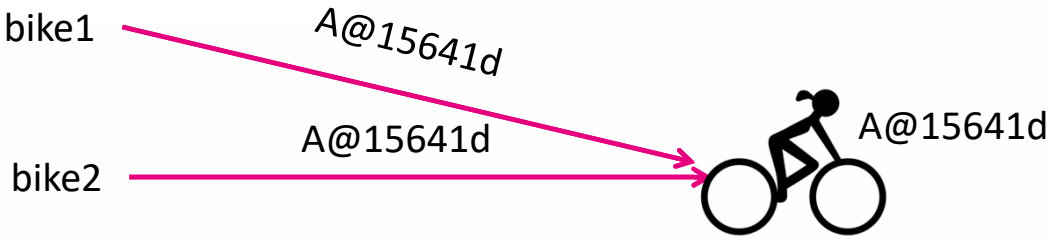
Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

bike1 now takes the same value as bike2

Objects stored at memory locations

A@93041f



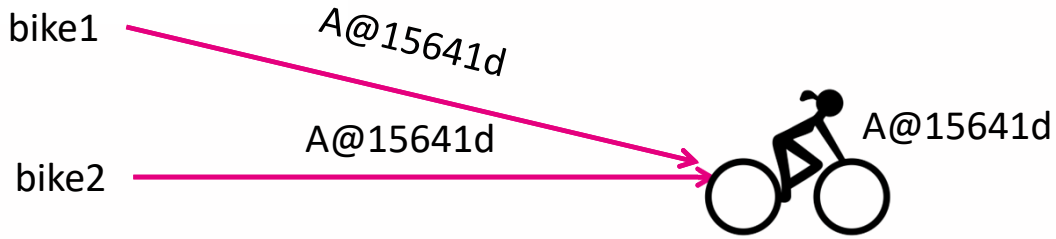
Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

bike1 now takes the same value as bike2

What happens to this object?

A@93041f

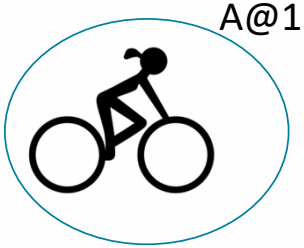


Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

What happens to both these object?

A@93041f



A@15641d

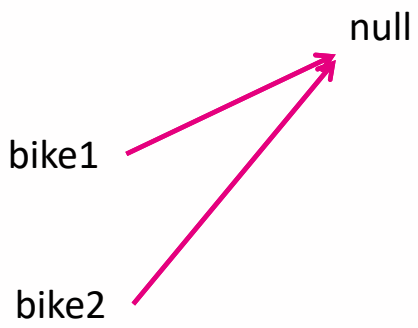
null

bike1

bike2

Memory Leak

This is what is called a memory leak.
In this case, you have two objects occupying memory and you have not deallocated them from memory
In fact, there is no way to deallocate them in Java!
So how do you deal with lost objects?



What happens to both these object?

Garbage Collector

- The Garbage collector is part of the JRE's memory management system
- It runs in the background keeping track of the live objects in a program and marking the rest as garbage
- The data in these marked areas are subsequently deleted, freeing up memory

Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Garbage Collector

null

bike1

bike2

Understanding References

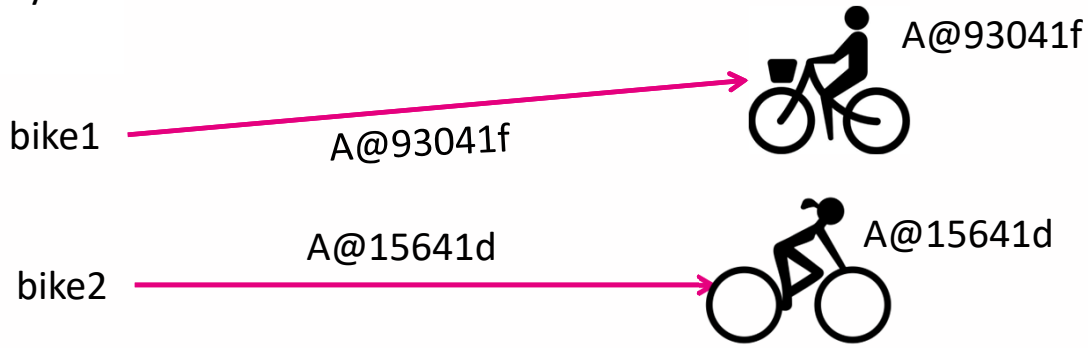
```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

The actual values of bike1, bike2 are memory locations

Garbage collector

A@93041f live

A@15641d live



Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

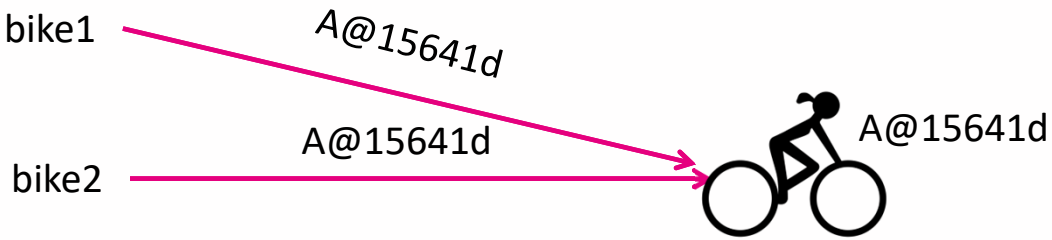
Bike1 now takes the same value as bike2

Garbage collector

A@93041f unreferenced

A@15641d live

A@93041f



Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Bike1 now takes the same value as bike2

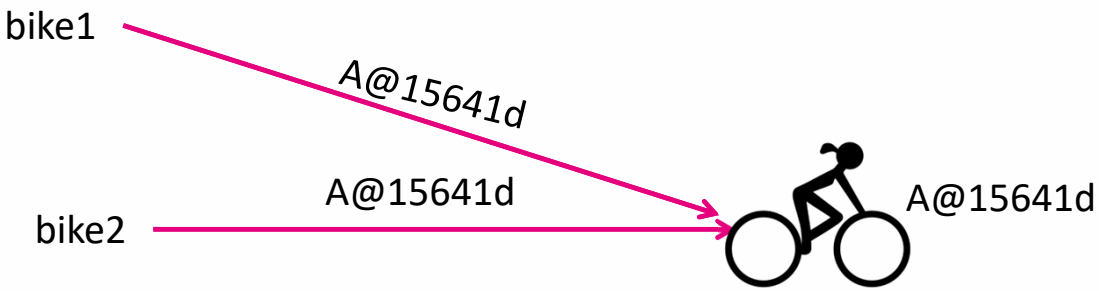
Garbage collector

A@93041f delete

A@15641d live

A@93041f

Yum, yum



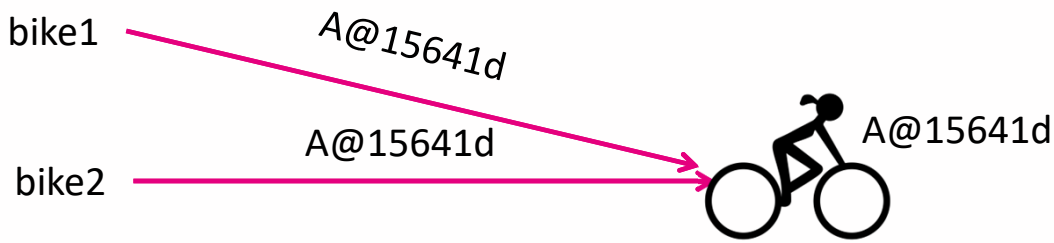
Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Bike1 now takes the same value as bike2

Garbage collector

A@15641d live



Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Garbage collector

A@15641d unreferenced

null

bike1

bike2

A@15641d

Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Garbage collector

A@15641d delete

null

A@15641d

bike1

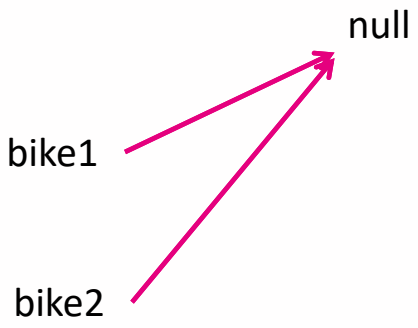
bike2

Understanding References

```
Bicycle bike1; //bike1 points to null
Bicycle bike2; // bike1 points to null;

bike1 = new Bicycle();
bike2 = new Bicycle();

bike1 = bike2;

bike1 = null;
bike2 = null;
```

Garbage collector

waiting…for its next unreferenced object



True or False?

The value of a variable in Java can be

1) A primitive
2) A reference value
3) An object

```
int x = 2;
Bicycle bike  = new Bicycle(1,2,3);
```

False

The value of a variable ✓ Java can be

1) A primitive ✓
2) A reference value
3) ~~An object~~ ✗

```
int x = 2;
Bicycle bike  = new Bicycle(1,2,3);
```

The value of a variable is **never** an object. However, it can take a reference value to an object

Assignment Steps

```
Car car = new Car("X7");
Engine engine = new Engine("DR9", 43);
car.add(engine);
Wheel wheel = new Wheel ("Wichelin15", 15);
car.add(wheel);
car.setFuel(100);
car.run();
car.getDistance();
```

Test-driven development

The code before is our **test**
It specifies the minimum we have to do to demonstrate the overall program works as per the problem specification
Once the code we have written outputs what we want, we can stop
This will be version 1 of our assignment

What we know

We have three classes: **Car, Engine and Wheel**

We know the **properties** of each class

We have composition relationships between them

- Car composed of Engine
- Engine composed of Wheel

We know that they have to create a few methods in each class so that objects can call each other in order for the program to deliver the functionality we require

Approach

Test-driven development = **incremental approach** to solving a problem
Incrementally create Stub classes and Stub methods so that your code compiles and runs at all times
To start with, it may run – but it may do nothing interesting.
Gradually we add functionality – making sure it compiles and runs
We keep doing this until we achieve our minimum criteria for success
In this case - we want to print out the distance achieved