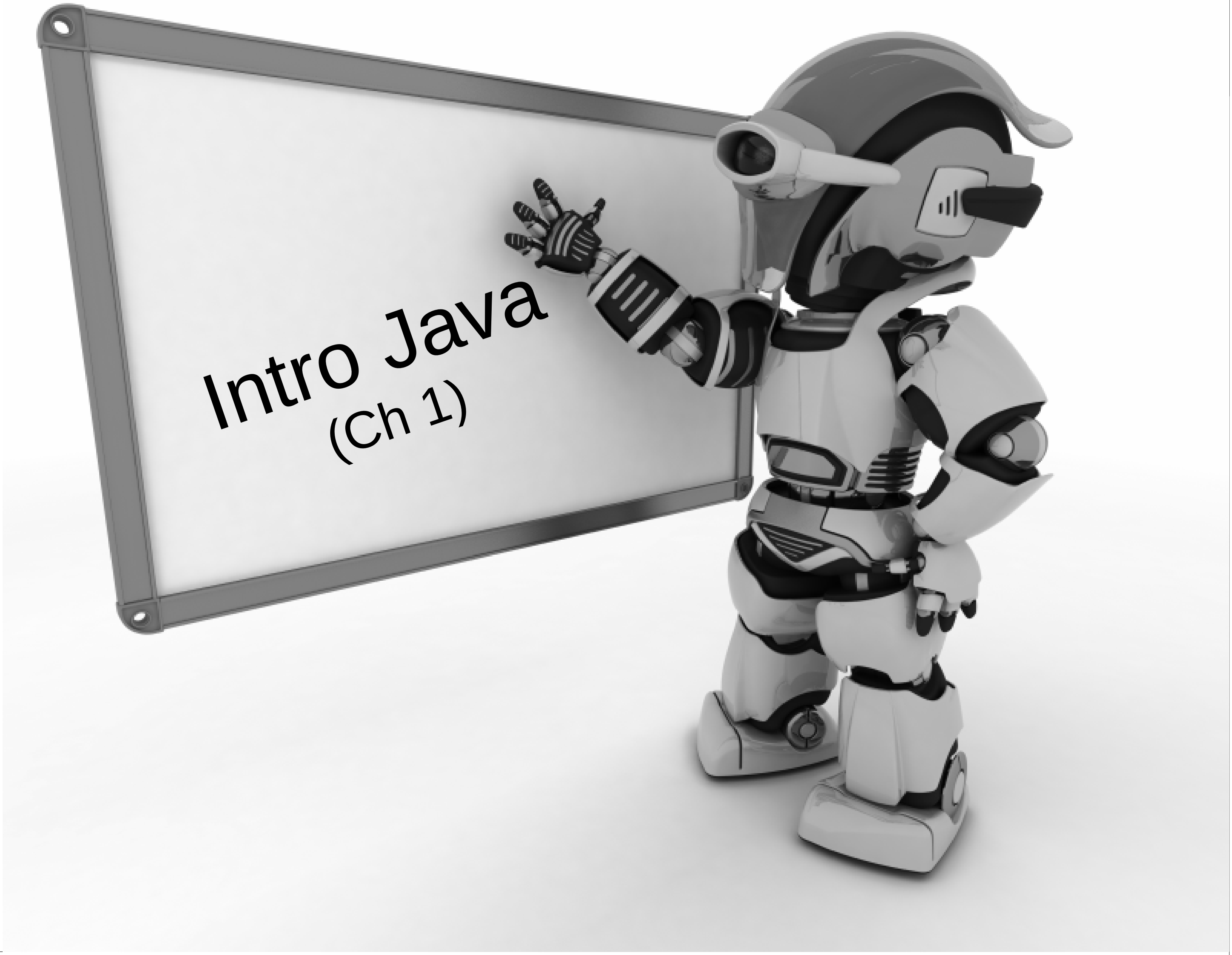


Intro Java
(Ch 1)

Hello Java World

Hello world

Execution starts in main()

public:	anyone can call
static:	don't need to instantiate
void:	no return value
main:	function run at start.

...

```
/**
 * Demonstrate use of main() and calling a static function.
 */
public class HelloWorld {

   public static void main(String[] args) {
      String courseName = "CMPT213";
      System.out.println("Hello " + courseName + " World!");
   }

}
```

System.out.println(): prints with linefeed

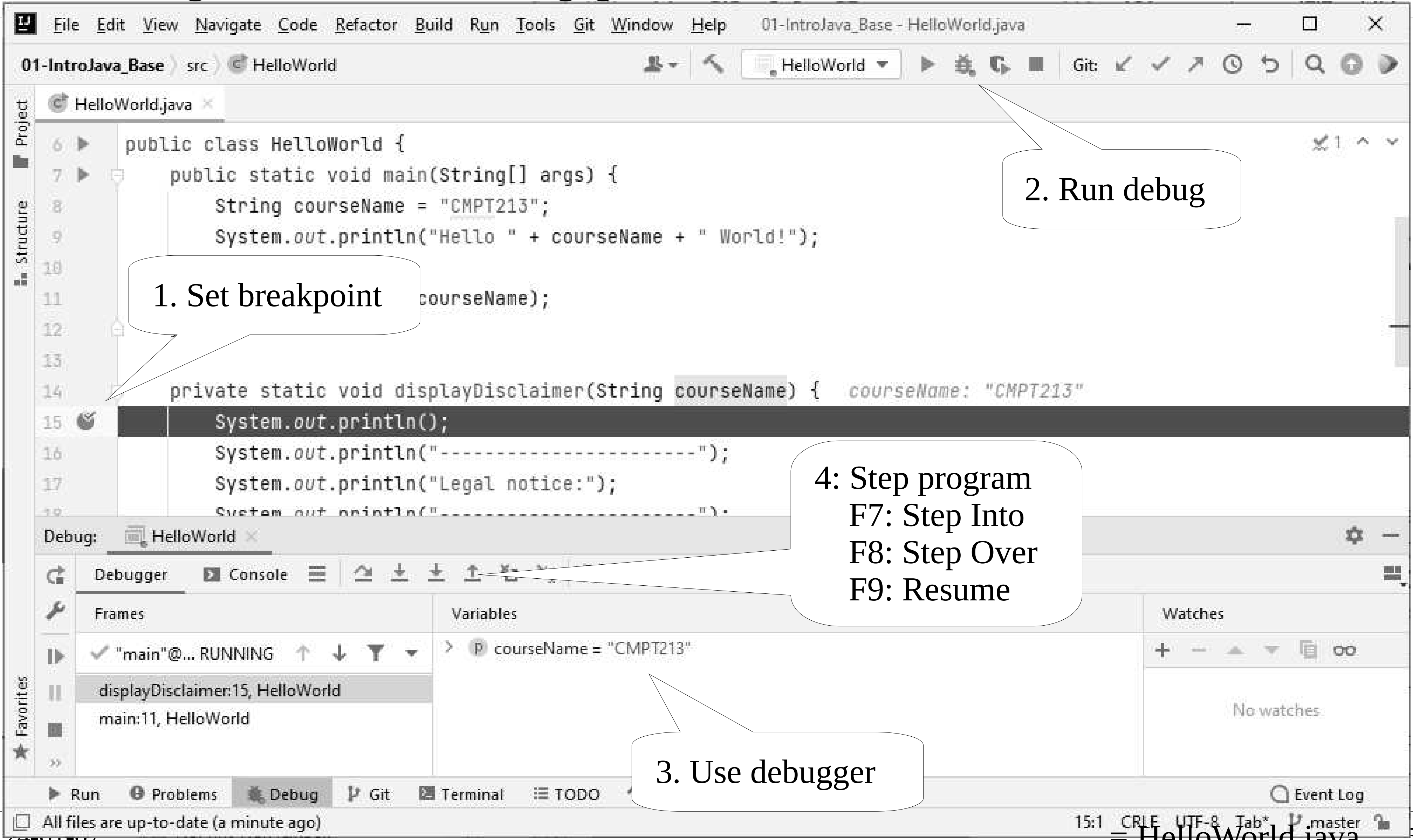
System:	class for accessing system data.
out:	field to write to console.
println():	method which write

Function

```
/**
 * Demonstrate use of main() and calling a static function.
 */
public class HelloWorld {

   public static void main(String[] args) {
      String courseName = "CMPT213";
      displayDisclaimer(courseName);
   }




   private static void displayDisclaimer(String courseName) {
      System.out.println();
      System.out.println("No warranty for " + courseName);
      System.out.println("or other \"persons\".");
   }

}
```

Create and call own functions.
- May call a function anywhere in the file (no need for function prototypes).

Integrated Debugger


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
01-IntroJava_Base - HelloWorld.java
01-IntroJava_Base src HelloWorld
HelloWorld
Git:
HelloWorld.java
Project
Structure
Favorites
Debug: HelloWorld
Debugger Console
Frames
Variables
Watches
Run Problems Debug Git Terminal TODO
Event Log

```
public class HelloWorld {
    public static void main(String[] args) {
        String courseName = "CMPT213";
        System.out.println("Hello " + courseName + " World!");

                            courseName);


    private static void displayDisclaimer(String courseName) {   courseName: "CMPT213"
        System.out.println();
        System.out.println("----------------------");
        System.out.println("Legal notice:");
```

```
"main"@... RUNNING
displayDisclaimer:15, HelloWorld
main:11, HelloWorld
```

```
courseName = "CMPT213"
```

No watches

All files are up-to-date (a minute ago)

15:1 CRLF UTF-8 Tab* master

1. Set breakpoint

2. Run debug

3. Use debugger

4: Step program
F7: Step Into
F8: Step Over
F9: Resume

Classes

- Class Name
  - Class HelloBob is in file HelloBob.java (case sensitive).
  - Constructor is same name as class; no return type.
  - *Convention*:..
- Field
  - a member variable or data stored by an object.
  - Called..
- Method
  - a member function of the class which may operate on fields.

Instantiating an object

```
public class GreetingsSelf {

   private String name;

   public GreetingsSelf(String name) {
      this.name = name;
   }

   public void setName(String name) {
      this.name = name;
   }


   public String getGreeting() {
      return "Hello der Java World, from " + name;
   }


   public static void main(String[] args) {
      GreetingsSelf greeter = new GreetingsSelf("CMPT 213");
      System.out.println(greeter.getGreeting());
   }
}
```

Private field

Constructor

Good practice:..

Instantiate new object.

= GreetingsSelf.java

One Name

- Use this to..
  - All objects are accessed by references.
  - References are like pointers but
    Java automatically dereferences when needed.
- Give each idea one name
  - Name field and constructor parameters the same.
  - Ex: name both numStudents, vs using each of:
    - studentCount
    - numStudents
    - n
    - numberStds

```
public class Course {
   private int numStudents;

   public Course(int numStudents) {
      this.numStudents = numStudents;
   }
}
```

Classes & Visibility

```
public class GreetingsWorld {
  private String name;

  public GreetingsWorld(String name) {
    this.name = name;
  }

  public String getGreeting() {
    return makeGreeting();
  }
  private String makeGreeting() {
    return "Hello Java World, from " + name
  }
}
```

Make all fields private whenever possible.

Public method can call private method

private:

..

public:

..

Classes & Visibility

```java
public class GreetingsWorld {
  private String name;
  public GreetingsWorld(String name) {...}
  public String getGreeting() {...}
  private String makeGreeting() {...}
}
```

```java
/**
 * Test the GreetingsWorld class
 * as a unit test.
 * Some code won't work!
 */

public class GreetingsWorldTest {
    private static final int TRIES = 5;

    public static void main(String[] args) {

        for (int i = 0; i < TRIES; i++) {
            GreetingsWorld greeter = new GreetingsWorld("Round " + i);
            String message = greeter.getGreeting();
            System.out.println("Name is: " + greeter.name);
            System.out.println("Name is: " + message);
            System.out.println("Name is: " + greeter.makeGreeting());
        }
    }
}
```

Which code won't work?

Cannot access private field or method from a different class!

..

Comments

- JavaDoc:
  commenting syntax used to generate documentation.
  - on a class: above a class to describe purpose of class
  - on a method: above a method (or field) to explain it
    - Suggest only using for API methods:
      stable interface and requires solid documentation for external users.
- Commenting Rules (this course):

  RULE 1:..

  RULE 2: Name fields, methods, and parameters well so ..

JavaDoc Example

```
/**
 * Helper class to compute useful properties of a right-triangle.
 * @author Brian Fraser
 *
 */
public class RightTriangle {
    /**
     * Compute the length of the hypotenuse of a right-triangle.
     * @param a Length of the first side (height); must be >=0.
     * @param b Length of the second side (base); must be >=0.
     * @return Length of hypotenuse.
     */
    public static double computeHypotenuse(double a, double b) {
        // ... Code omitted.
    }
}
```

Our code won't (usually) have method comments though!

Primitive Types

- Primitive Types..
- char is..
  2 bytes per character
  - Escape sequences:
    '\\', '\n', '\t', '\''
- boolean holds value..
- Everything else is an object reference

```
/**
 * Show the different primitive types.
 */
public class PrimitiveTypeDemo {
    public static void main(String[] args) {
        byte  next8Bits = 0x30;
        short dayOfMonth = 13;
        char  firstLetter = 'A';
        int   age = 42;   // 32 bit signed
        long  numberAtoms = 25000000000000L;
                          // 64 bit signed

        float  weight = 150.15F;
        double timeSinceStart = 1.1;

        boolean isAwesome = true;
    }
}
```

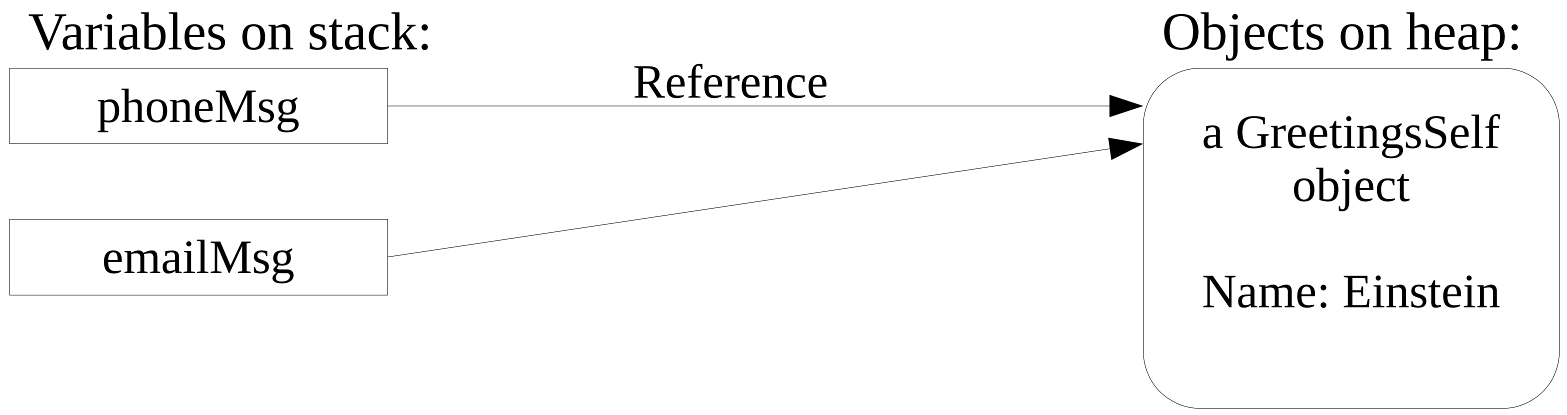
Type conversion

- ..
  - Converting from smaller type to larger: widening conversion
  - OK to do implicitly.
    double weight = 200;
- ..
  - Converting from a larger type to a smaller one.
  - Must cast because can lose data: narrowing conversion
    int height = (int) 10.99;
    float length = (float) 12.0;   // Why needed?
- Constants
    final int MAX_LENGTH = 100;
  - RULE:..
    0, 1, (& sometimes -1 or 2) are often non-magical.

Multiple Object Reference

- = on an ***object*** **reference**..
- Example

```
GreetingsSelf phoneMsg = new GreetingsSelf("Einstein");
GreetingsSelf emailMsg = phoneMsg;

emailMsg.setName("Albert");
```



- Automatic Garbage Collection
  - Objects with no references to them are automatically deleted.

Control Structures

- Same control structures as C/C++.
  - Note boolean is not an int, so if (j = 10) { ... } is a..

```
public static int demoControlStructures() {
    final int MAX = 10;
    boolean isHappy = true;

    for (long i = 0; i < MAX; i++) {

        int j = (int) i;
        while (j < MAX) {

            if (j == i + 1 && !isHappy) {
                break;
            } else {
                isHappy = false;
                j++;
            }
        }
    }
    return 0;
}
```

Static,
Exceptions,
& Debuging

Static

- Static method
  - Can be called on the class (no object required).
  - Also called..
- Static field
  - Shared by all instances of the class.
  - Also called..
  - Often used for constants:
    public static final int DAYS_PER_WEEK = 7;
- Static local
  - Not supported in Java.

Static: What fails to compile?

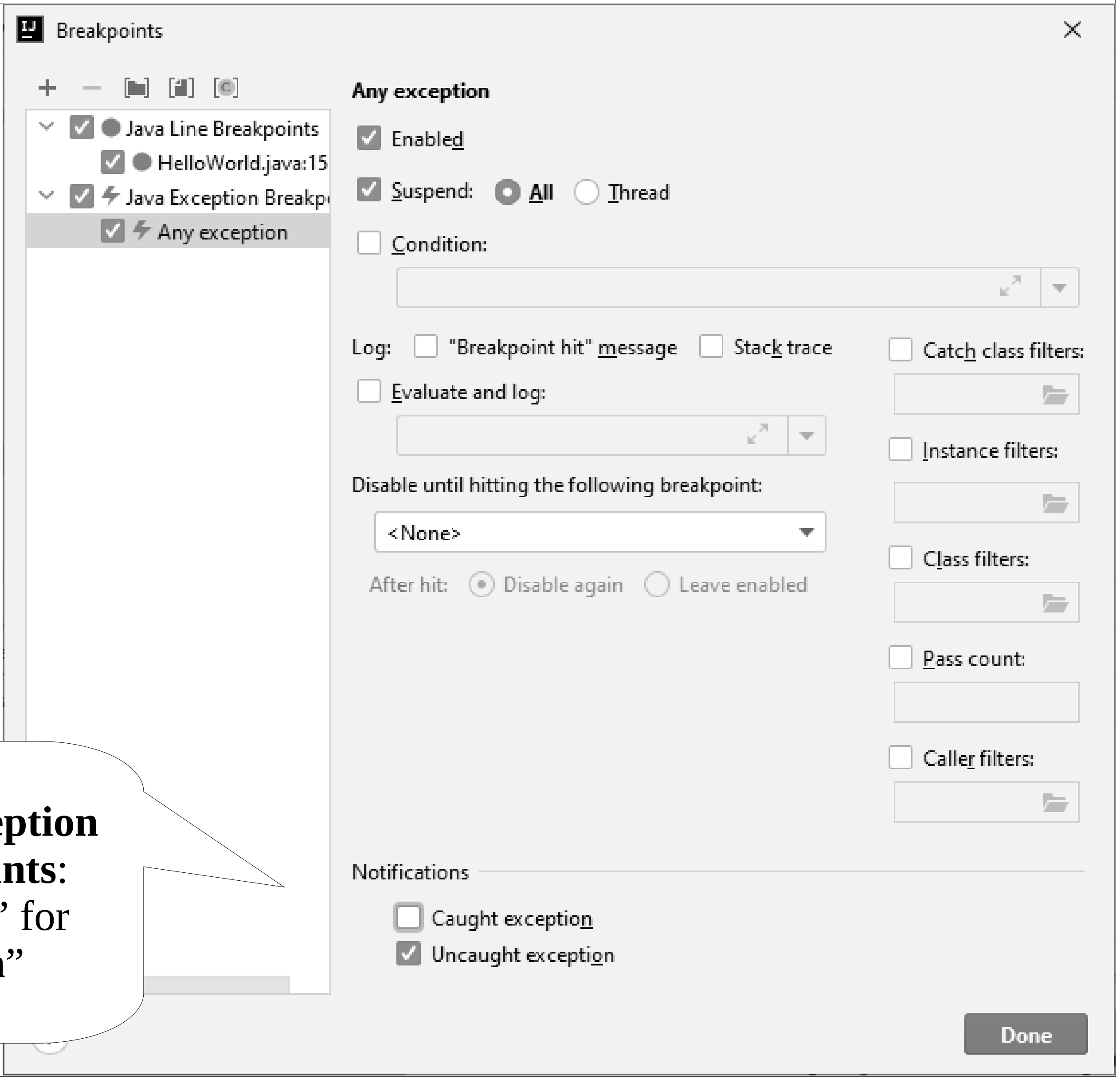
```
public class StaticFun {
    public static final int TARGET_NUM_HATS = 10;
    private static int countNumMade = 0;
    private int favNum = 0;
    public static void main(String[] args) {

        // WHICH OF THESE 4 LINES GIVES A COMPILE TIME ERROR?
        changeFavNum(42);
        displayInfo();
        favNum = 10;
        countNumMade = 9;

    }
    private void changeFavNum(int i) {
        favNum = TARGET_NUM_HATS + i;
        displayInfo();
    }
    private static void displayInfo() {
        System.out.println("TARGET_NUM_HATTS: " + TARGET_NUM_HATS);
        System.out.println("countNumMade:     " + countNumMade);
        System.out.println("favNum:           " + favNum);
    }
}
```

Exceptions

- Java..   on some errors
- Examples:
  - HelloWorld c;
    c.xyz();   // Throws null pointer ex.
  - int oops = 10 / 0;   // Throws div. zero ex.
  - // Throw your own, they are objects.
    throw new RuntimeException("Busted!");

Debugging Exceptions

- Exercise
  - Debug Rectangle.java with IntelliJ
  - Use debug, breakpoints, step over/into, watch variable
  - Input: 10, -1



Any exception

[✓] Enabled

[✓] Suspend: (● All) ( ) Thread

[ ] Condition:

Log: [ ] "Breakpoint hit" message [ ] Stack trace

[ ] Evaluate and log:

Disable until hitting the following breakpoint: <None>

After hit: (● Disable again) ( ) Leave enabled

[ ] Catch class filters:

[ ] Instance filters:

[ ] Class filters:

[ ] Pass count:

[ ] Caller filters:

Notifications

[ ] Caught exception

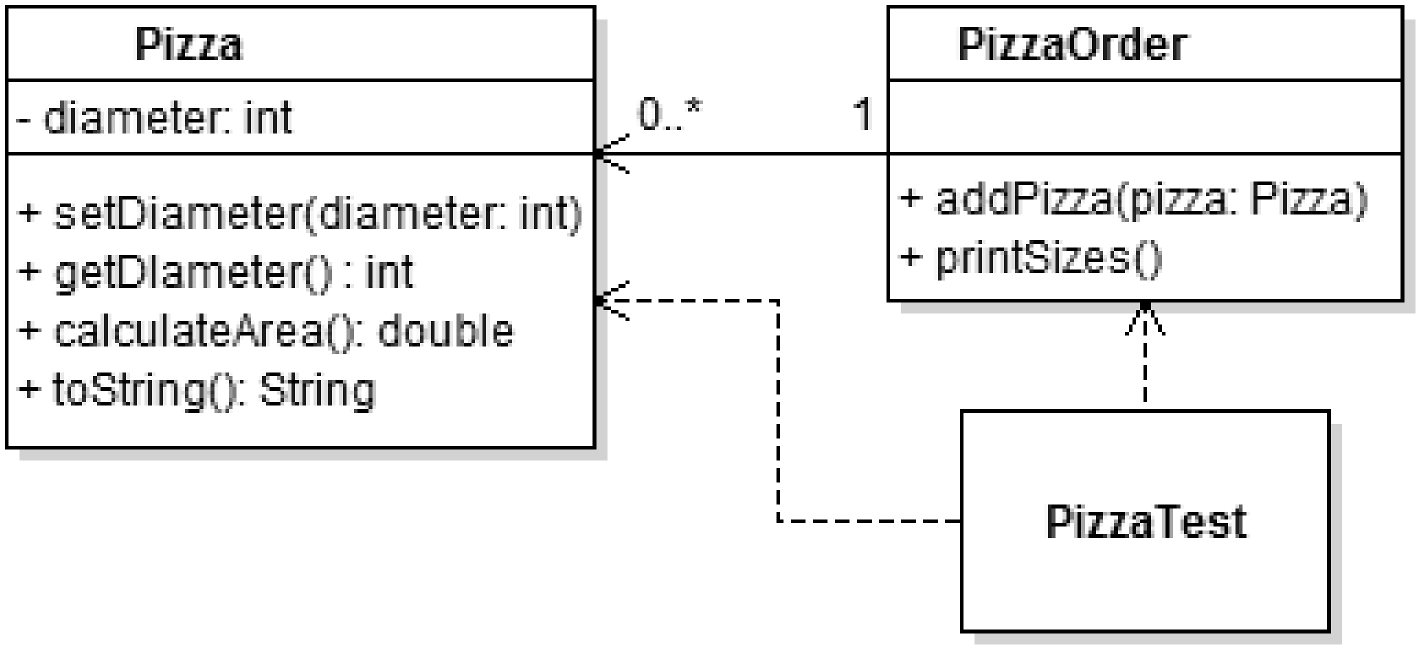
[✓] Uncaught exception

Break on Uncaught Exception
Run --> View Breakpoints: enable "**Any Exception**" for "**Uncaught exception**"

Pizza Class Example
(package, Math, toString(), pass by...,
array, ArrayList, for each)

UML

- We will create the following classes in this section of the slides.



Packages

- Java organizes code into packages.
  Ex: ca.cmpt213.as1 or com.ibm.db2.query
  - Set the package:
    package ca.sfu.webreg.login;
  - Save .java files into:
    src\ca\sfu\webreg\login\...
  - Can use code from a different package:
    import ca.sfu.webreg.login;
    or
    import ca.sfu.webreg.*;

Pizza (step 1)

- Create a new Java project in IDE (IntelliJ).
- Create a Pizza class inside a new package.
- Pizza Class features
  - Store the diameter as an int; use constructor to set.
  - Create accessors and mutators for diameter.
    - Do we need a mutator?
- Create a PizzaTest class
  - Give it a main().
  - Create new function to test Pizza so far.

Math

- Math class has useful static fields and methods
  - Math.PI
  - Math.pow()
  - Math.ceil(), Math.floor(), Math.round()
  - Math.abs()
  - Math.min(), Math.max(),
  - Math.signum(x) // 1.0 if x>0, -1.0 if x<0, 0 if 0.
  - Math.random()
  - Math.toDegrees(), Math.toRadians()
- Pizza Example
  - Create & test method to get the pizza's area.

toString()

- All Java objects have a toString() method
  - All classes inherit from Object, which implements toString()
- Returns a String object which..
  - Used for **debugging**,..
  - Recommended format:

```
@Override
public String toString() {
   return getClass().getName()
      + " [daField1=" + daField2
      + ", daField2=" + daField2 + "]";
}
```

@Override Annotation: method overrides a base class's method. (optional)

getClass().getName() returns class name of current object.

..

- Pizza: Implement meaningful toString();

Pass by value

- Java uses pass by value
  - Passing a primitive type passes its value.
  - Passing an object passes (by value)..
- What this means
  - When passed a primitive type, changes inside a method have no effect outside the method.
  - When passed an object, you *can* modify its state.
  - You *cannot* change..

Passing Example

```
void demoPassByValue() {
    int myFavNum = 42;
    changeNumber(myFavNum);
    System.out.println("Number: " + myFavNum);

    Pizza myPizza = new Pizza(20);
    modifyPizza(myPizza);
    System.out.println("Area (1): " + myPizza.calculateArea());
    changeWhichPizza(myPizza);
    System.out.println("Area (2): " + myPizza.calculateArea());
}

void changeNumber(int x) {
    x = 0;
}

void modifyPizza(Pizza pizza) {
    pizza.setDiameter(2);
}

void changeWhichPizza(Pizza pizza) {
    pizza = new Pizza(10);
}
```

What is the effect of each method?

Arrays

- Arrays have a fixed size when created:

```
int[] ages = new int[10];
Hat[] hats = new Hat[2];
```

  - 0 indexed.
  - Bounds checked!

```
int size = ages.length;     // it’s a field, not size() method
int first = ages[0];
int oops = ages[size];      // throws exception; why?
```

- Demo: Show PizzaOrder
  - store up to *N* Pizzas (argument to constructor)
  - implement Pizza.add(Pizza) and Pizza.printSizes()
  - Test with PizzaTest

for-each loop

- Java includes the “enhanced for loop”
  - Previously
    ```
    for (int i = 0; i < hats.length; i++) {
       Hat hat = hats[i];
       System.out.println(“Hat: ” + hat.getColour());
    }
    ```
  - Enhanced Loop
    ```
    for (Hat hat: hats) {
       System.out.println(“Hat: ” + hat.getColour());
    }
    ```
  - No need to manage loop index (can't get it wrong!)
    ..

List and ArrayList

- Generic: works with..
- Java includes many generic Collections.
  - ArrayList implements the List interface and is backed by an array (fast), and dynamically resizes.

```
List<Hat> hats = new ArrayList<>();
hats.add(new Hat("Blue"));
for(Hat hat: hats) {
     ...
}
```

Don't need to put <Hat>, the type, because already specified on left-side.

- Collections only store objects...
  - To store primitives, use built in.. Integer, Long, Double, etc.
- Demo: Change PizzaOrder to ArrayList.

“Strings”

Strings

- String Class
  - Stores strings in Unicode: 2 bytes per character.

```
String msg = "Hello";
char first = msg.charAt(0);
```

- String literals are..

```
int length = "Hello".length();
```

- Many methods on String
  -
```
.length(),          .contains(...),
.endsWith(...),     .isEmpty(),
.replace(...),      .split(...),
.toLowerCase(),     .trim()
```

Comparing Strings

- Compare strings using..

```
String password = getDaUsersPassword();
if (password.equals(“12345”)) {
    System.out.println(“The air-shield opens.”);
}
```

- Don't use ==
  - == compares the..

```
if (password == yourGuess) {
    String msg = “Wow! The program stores the ”
        + “password and your guess at the same ”
        + “memory location! Crazy!”;
    System.out.println(msg);
}
```

Immutable

- Strings are Immutable
  Once created,..
  - To “change” a string,..
- Example

```
String msg = "H";
msg = msg + "i";
msg += '!';
int count = msg.length();
```

Creates 3 strings;
2 for garbage collection:..

- Java does not support overloaded operators in general, except for + and += on Strings.
  - String still immutable, even with +=

String Demo

```
static void demoStringConcat() {
    String guess1 = "hello " + 42;
    String guess2 = "hello " + 4 + 2;
    String guess3 = 42 + "hello";
    String guess4 = 4 + 2 + "hello";
    String guess5 = new Integer(42).toString();
}
```

What does each String hold?

```
static void demoStringToNumber() {
    String myInput = "42";
    int theValue = Integer.parseInt(myInput);
```

Also have:
Double.parseDouble(...)
Boolean.parseBoolean(...)
Long.parseLong(...)

```
    // Current date/time to string
    Date now = new Date();
    String msg = "Currently " + now;
    System.out.println(msg);
```

Date.toString() gives:

```
Thu Jan 16 13:49:46 PST 2014
```

Date in java.util.Date

```
    // Demo bad conversion
    int oops = Integer.parseInt("Oops");
}
```

Throws NumberFormatException

Keyboard Input

Scanner

- Scanner class
  - Keyboard input done via the Scanner class (in java.util.Scanner)
- Example

```
// Setup
Scanner daScanner = ..

// Use:
System.out.println("Enter your age: ");
int age = ..
```

Scanner for bad type

- Reading wrong type of data..

- Example
```
int diameter = scanner.nextInt(); // but Type “hi!”
```
- Two ways to avoid this exception:

```
int diameter = 0;
try {
    diameter = scanner.nextInt();
} catch (InputMismatchException ex){
    System.out.println("int only!");
}
```

```
int diameter = 0;
if (scanner.hasNextInt()) {
    diameter = scanner.nextInt();
} else {
    System.out.println("int only!");
}
```

Scanning Line Feeds

- Read a line with .nextLine()

  String fullLine = myScanner.nextLine();

- Linefeed Complication
  - Scanner.nextInt()..

    like a linefeed.

```
System.out.print("Enter age: ");
int age = scanner.nextInt();

System.out.print("Enter name: ");


String name = scanner.nextLine();
System.out.println("Hello " + name
            + " of age " + age);
```

Closing Scanner

- Java does garbage collection on unused objects, but some objects..
  - Example: File, network socket, input stream.
  - Must explicitly close these objects or suffer a..

- However, System.in need not be closed
  - It is provided by the OS, so don't close a Scanner created from System.in.
  - Other Scanners must be closed (such as for files).
  - Can hide the warning with annotation: @SupressWarnings("resource")

Text Files

Java Classes for Text Files

- File(filePath)
  - Represents a single file on disk (by path).
  - Package: java.io.File
- Scanner(File)
  - Does reading, use .hasNextInt() .nextInt()
  - Package: java.util.Scanner
- PrintWriter(File)
  - Does writing, use .println()
  - Package: java.io.PrintWriter
  - Use PrintWriter for a file or the screen:

    PrintWriter myWriter = new PrintWriter(System.out);

Write to file

Create a File object for target file.

Catch exception: FileNotFoundException

Write to the file via the PrintWriter

Close the PrintWriter

```
File targetFile =
   new File("C:/dos/run/test.txt");

try {
   PrintWriter writer =
      new PrintWriter(targetFile);

   writer.println("Run DOS run!");
   writer.println("Ok.. old joke...");

   writer.close();
} catch (FileNotFoundException e) {
   // TODO: Handle this!
   e.printStackTrace();
}
```

Never squelch an exception:
..
- Log (or print) an error
- Rethrow: throw new RuntimeException(e)

Read from file

Create a File object for source file.

Open a new Scanner. Catch exception: FileNotFoundException

Read all data from file via Scanner

Close the Scanner

```
File sourceFile =
   new File("C:/dos/run/test.txt");

try {
   Scanner scanner =
      new Scanner(sourceFile);

   while (scanner.hasNextLine()) {
      String text = scanner.nextLine();
      System.out.println("Read:" + text);
   }

   scanner.close();
} catch (FileNotFoundException e) {
   // TODO: Do something better here?
   e.printStackTrace();
}
```

Static Factory Method

- Static Factory Method
  - A..
  - Like a constructor, but more flexible:
    can give it a..
  - A common..
- Example
  - In Pizza class:

```
public static Pizza makePizzaFromFile(File file) {
    // Open file and read in values
    // Create new Pizza object
    // Return the Pizza
}
```

When is your code done?

Coding Standards

Clean Code

- Correct Code
  - Implements the requirements.
  - Has no (few) bugs.
- Clean Code
  -
  - Conforms to..
  -
  -
- Professionals write clean code.

Coding Standard

- Course (and most companies) has a coding standard (See web page)
  - Your code *must* conform to this style guide.
  - Each assignment may mention some specifics.

- Activity
  - Read Coding Standard.
  - Go through the Person class and clean it up.

Summary

- Classes: public, private, static, constructor, package, JavaDocs, toString()
- Primitive types, type conversion, wrappers
- Arrays, ArrayList, for-each
- String: Immutable class for working with all strings.
- Scanner for input (file or keyboard)
- PrinterWriter for output to file
- Coding standard enforced for clean code.