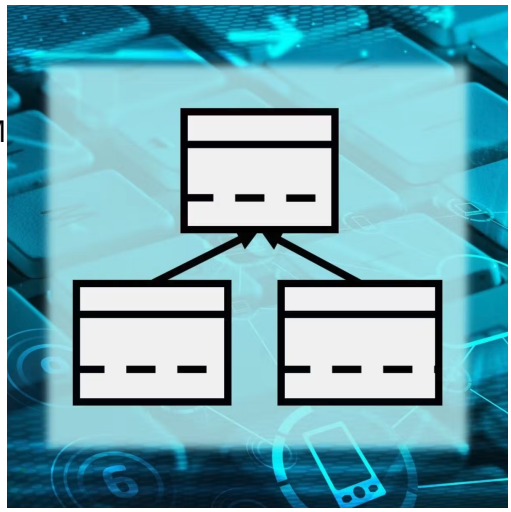
About this Course



Instructors: R. Jordan Crouser (https://jcrouser.github.io) and Johanna Brewer (http://deadroxy.com/)

Course Meets: TTh 10:50-12:05PM EST (Section 01) / 1:20- 2:35PM EST (Section 02)

Location: Bass 204 (https://maps.app.goo.gl/QHGZwnCdatHUXj959)

Jordan's Office Hours: Tuesdays 8:30-10am and Fridays 8:30-10am in Bass 105 (https://goo.gl/maps/h6MQbQNM38bELqpK6)

Johanna's Office Hours: TBD

TA Hours: Sundays 1-3pm EST, Sundays through Wednesdays 7-9pm EST in Ford Hall

---

Course Description:

This course emphasizes computational problem-solving using a typed object-oriented programming (OOP). Students will learn core computer science principles including: control flow, functions, classes, objects, methods, encapsulation and information-hiding, specification, recursion, debugging, unit testing, version control, using libraries and writing code in multiple files. Abstract data types and simple data structures will be used to illustrate concepts of OOP and solve computational problems through regular programming assignments (in Java and Python).

This course assumes prior programming experience including a basic understanding of branching (if-statements), iteration (loops), functions and simple data types (integers, strings, lists/arrays). *Prerequisites: CSC 110 or equivalent. Cannot be taken concurrently with CSC 110.*

Learning Goals:

Upon successful completion of this course, students will be able to:

- Use object-oriented techniques to write programs.
- Develop classes and interfaces with overloaded methods.
- Use inheritance, polymorphism, and abstract classes.
- Design and implement robust, reusable, human-friendly programs (including the development of unit tests and custom exception classes).

Syllabus

Welcome to CSC120: Object-Oriented Programming! Below is some important information about how the course will run, as well as some specific "expectation-setting" details about what we will (and won't!) cover.

Course Format

This is a programming-intensive course, designed to help early-stage programmers transition into developing software engineers. While programming can be a truly interesting, challenging and rewarding intellectual activity, it is almost impossible to learn it without **a lot of practice**. This takes time, especially for debugging a troublesome program that isn't quite working right yet.

For this reason, the course features a **SEMI-FLIPPED CLASSROOM**. This means occasionally you'll be asked to watch short prerecorded introductions for certain topics and complete some activities *before class*, and we'll spend our time together in class writing code, starting assignments, and applying the concepts from lecture in real-world contexts. However, this model only works if you *commit to the process*… and if this is your first experience in a semi-flipped classroom, it might feel a little overwhelming. Here are some useful tips to help orient you to this model (adapted from UC Boulder's Succeeding in a Flipped Classroom (https://www.colorado.edu/assett/2017/03/21/succeeding-flipped-classroom-march-2017-tip):

1. **Expect work to be continuous** Rather than cramming for large exams that happen *after* classes, you are required to keep up with frequent assignments, readings, and videos that are due *before* class. One way to set yourself up for success is to get into the habit of completing your pre-class tasks at the beginning of the semester. Keep up with Discord and Moodle for updates and assignments.

2. **Show up present and prepared** Be present in class, physically and mentally, to the best of your ability. Be ready engage in class discussions and activities. Ask questions to clarify your understanding of concepts, offer your own perspective, and try not to be afraid of giving "wrong" answers – misconceptions and *false starts are a normal, healthy part of learning* (and we professors guarantee we'll make plenty of our own mistakes for you to catch!)

3. **Use prerecorded lectures and readings to your advantage** While watching prerecorded mini-lectures may seem like a pain (or just a way for your Prof. to get out of lecturing :-P), they provide opportunities to learn at your own pace as well as get the perspective of other instructors (as many of the videos we will use to lay the foundation for this course come from beyond Smith). If you benefit from rewatching videos or pausing to try something out on your own, please do! Many students find it helpful to take notes while watching prerecorded lectures, just like they would in a traditional class.

4. **Your education is everyone's priority** If you find yourself spending inordinate time debugging your programs without really making progress, it's time to reach out – *before you fall behind*, or it affects your work in other courses. Your professors and TAs are eager to help you find strategies

that work for you, and that enable you to reach your goals in the course. In addition to us, the college has many resources for academic success (https://www.smith.edu/academics/resources-and-services), all available at no cost.

5. **Give honest feedback** All teaching is improvable, and so we'll ask for your feedback often. We want to know what is going well and what is tough. Being open and constructive will help us to improve the class, both for future students and for *you*.

Course Materials

There is **no required textbook** for this course: all required readings and videos will be made available through Moodle. However, the following recommended textbook is a good choice for those who find them useful as references. It is available for free online, or you can purchase a hard copy from your favorite bookstore:

- Think Java: How to Think Like a Computer Scientist, 2nd Ed. (https://greenteapress.com/wp/think-java-2e/) (Allen Downey and Chris Mayfield.), O'Reilly, 2019.

  Think Java is a hands-on introduction to computer science and programming used by many universities and high schools around the world. Its conciseness, emphasis on vocabulary, and informal tone make it particularly appealing for readers with little or no experience. The book starts with the most basic programming concepts and gradually works its way to advanced object-oriented techniques. In this fully updated and expanded edition, authors Allen Downey and Chris Mayfield introduce programming as a means for solving interesting problems. Each chapter presents material for one week of a college course and includes exercises to help you practice what you've learned.

If you need help covering the cost of textbooks or other academic supplies (for this or any of your courses!) please fill out the Academic Funding Application found at socialnetwork.smith.edu/forms (socialnetwork.smith.edu/forms).

What We Will Cover

This course builds a strong foundation in object-oriented programming. It is designed **for CSC majors who have completed a first course in programming**. If this isn't you, don't worry! You're still very much welcome here. Just be mindful that some self-study before the course starts may be necessary (see What We WON'T Cover).

Here's a selection of topics we will cover:

- Object-oriented programming in Java
- Pointers, references, and indirection
- Encapsulation
- Inheritance
- Composition
- Polymorphism
- Abstraction

What We **WON'T** Cover

This course assumes that you have a solid foundation in computational thinking, either through successful completion of CSC110, high school curriculum at at AP or IB level, or dedicated self-study. As such, we expect you to come to class with a solid understanding of the following topics (which we may recap briefly to establish common language, but which won't be covered in substantial detail):

- Conditionals, math operators, logical operators
- Loops and iteration
- Variables and data types
- Strings
- Lists
- Dictionaries
- Reading and writing to file storage
- User inputs and print formatting
- Functions
- Basic concept of an algorithm
- Understanding abstract components (e.g., input, output, storage, computation)
- Awareness of physical components (e.g., hard drive, RAM, CPU, keyboard)

Proficiency with various computer environments and infrastructure will also come in handy (though we will go over some of this in class together). Check out the following links to supplemental online courses through LinkedIn Learning if you'd like some additional support, which are available for free to all members of the Smith community:

- Programming Foundations: Fundamentals (https://www.linkedin.com/learning/programming-foundations-fundamentals-3)
  Been awhile since you took 111? This 2hr crash course will remind you of the key points. Taught in Python.
- Learning Java (https://www.linkedin.com/learning/learning-java-4/)
  Worried about making the leap from python to Java? This course is essentially a do-over of 110, but in Java instead of python.
- Java Code Challenges (https://www.linkedin.com/learning/java-code-challenges)
  Partway through the semester and you still feel like you just don't quite "get" Java? Try these challenges to build your confidence!

Communication

All written communication regarding this course will take place via Discord (https://discord.com) (a cloud-based communication platform that supports text, voice, and video). This includes:

- **#announcements**: Important notices about class times, deadlines, office hours, etc.
- **#general**: Introductions and discussion of course-related material, off-topic ideas, etc.
- **#questions**: Channel for all non-personal questions. *Don't be shy if you are puzzled, someone else probably is too!*
- **DMs**: Message your professors directly for matters that require individual communication.

Even if you're not used to Discord at first, it's not too hard to learn. The advantages of having all course communications in one place are compelling. Use Discord!

Assessment

There are three forms of assessment in this course:

- Weekly programming homework assignments
- Two written skill checks (one midterm, one final)
- One final programming project

Homework, Labs, and Lateness Policy

Weekly programming assignments will be introduced in class on Thursdays (sneak peek on Wednesday nights), and we will work together to get started on them during class time. Final submissions will be due Wednesdays at 11:59PM EST. Because our lives and learning do not always go as planned, every student will be able grant themselves extensions on homework assignments. Read details on how to request an extension here (late-policy.html).

Collaboration and Academic Integrity

Programming is more fun in groups! Students are strongly encouraged to form study groups and to collaborate in solving the assignments. Please ensure that all work you submit is ultimately the product of your own understanding rather than anyone else's. You may consult online or print references on all assignments and labs. Standard language references showing syntax, usage, documentation, etc. need not be cited; nor does the course textbook. All other resources must be cited as described below.

The following information is required for all submitted work:

1. The names of all collaborating students be listed at the top of the submission. If you worked alone, please state: "*I did not collaborate with anyone on this assignment.*"
2. A **References** section, with in-line citations to any external resources you used. Citations should include page numbers (if a printed resource) or a direct URL (if an online resource). If you did not use any resources in completing the assignment, please state: "*I did not utilize any external resources in completing this assignment.*" If you include a fragment of code from any source, you should also credit that source with a comment directly in the code.

Use of AI Code-Completion with Attribution

In this course, students are permitted to utilize AI-powered code-completion tools, provided you do so while adhering to responsible and ethical practices. These tools can provide valuable assistance in enhancing your programming efficiency and proficiency. You are permitted to incorporate AI code-completion suggestions into your coding assignments and projects, as long as proper attribution is given. ***Generative AI (e.g. ChatGPT or similar) cannot be used for written reflections in CSC120.***

Guidelines for Using AI Code-Completion:

1. **Attribution:** We'll treat AI code-completion tools as "collaborators" in this class. Whenever you get help with your programming tasks, it is crucial to provide clear and transparent attribution. Include a comment or annotation in your code specifying that certain sections were generated with the help of an AI code-completion tool.
2. **Originality:** While AI code-completion can offer valuable insights and suggestions, it is important that the final code reflects *your* understanding of the material. For this reason, you should avoid copying generated code without understanding what it does; instead, use it as a reference to enhance your own programming skills.
3. **Learning Opportunity:** View AI code-completion as a supplementary learning resource. Take the time to assess the suggestions provided by the tool and compare them to your own coding decisions. This process can contribute to a deeper understanding of the programming concepts we cover.
4. **Honor Code:** Always prioritize academic integrity. Plagiarism, which includes submitting someone else's work (including AI-generated content) without proper attribution, is a violation of our community's ethical standards and course policy.
5. **Discussion and Collaboration:** While using AI code-completion, feel free to engage in discussions with peers and instructors about the generated code and how it aligns with course concepts. Collaborative learning and constructive feedback can enrich the educational experience.
6. **Diverse Approaches:** Keep in mind that there are generally many "right" ways to solve a programming problem. AI-generated suggestions might present one approach, but exploring alternative solutions on your own or through discussions is highly encouraged.
7. **Human Power:** All AI is developed by other humans and trained on data generated by millions of our peers. Generative AI regurgitates and remixes existing information. Do not be fooled into thinking it "knows" more than you.

Remember: the primary goal of this course is to enhance your programming skills and understanding of the subject matter. Utilizing AI code-completion tools with attribution can support this goal, but the responsibility lies with you to ensure that your work reflects your own efforts and comprehension.

Grading

Category	Percentage
Homework	60%
Skill Checks	20%
Final Project	15%
Participation and Engagement	5%

Note that the final grade is based on our evaluation of your work, and every effort will be made to communicate expectations in advance through detailed rubrics. Although the grade will be largely based on the percentages shown above, we reserve the right to award extra credit for excellent work

and out-of-the-box thinking. For example, while “Participation and Engagement” will look primarily at day-to-day engagement, we will also take note of contributions both in and out of class which demonstrate intellectual curiosity or clear understanding of a topic, as well as comments which help others to learn a difficult concept.

Accessibility & Accommodations

We aim to make this course accessible to all and welcome feedback about changes we can make to meet that goal. If you encounter barriers to participation in this or any other course, please register with the Disability Services Office (https://www.smith.edu/about-smith/disability-services/register) to request support and accommodations.

Comfy Class Policy

Everyone is welcome to make themselves comfortable in our classroom and asked to be respectful of one another. When you are communicating, please practice active listening by focusing on understanding what others are expressing rather than thinking of how you will respond. Additionally, keep in mind that our wide array of individual backgrounds shape our unique perspectives, so please respect one another when we have sincere differences of opinion.

You may bring beverages or snacks, but please use closed containers to avoid spills and keep messy foods away from computers. Everyone is free to use concentration accommodations like fidget toys, knitting, doodling, moving around, or sitting on the floor; just be mindful your focus does not disrupt others. Parents and caregivers may bring their babies and children to class whenever necessary. Learners of all stages are invited to join us.

Acknowledgement

Some of the materials used in this course are derived from lectures, notes, or similar courses taught at other institutions. Appropriate references will be included on all such material.

Schedule

The following schedule is subject to change based on the progress of the class. Please check back frequently!

Date	Day	Topic	Assignment Out
1/25	Thur.	Introduction	A1: Real-World Objects
1/30	Tues.	Getting started with VSCode + git	
2/1	Thur.	Object-Oriented Thinking	A2: Object-ification
2/6	Tues.	Software Engineering Skill: Translating to/from Pseudocode	
2/8	Thur.	Introduction to Java	A3: Our First Java Class
2/13	Tues.	Encapsulation in Java	
2/15	Thur.	Memory Models	A4: Trace the Execution
2/20	Tues.	Exceptions	
2/22	Thur.	No Class - RALLY DAY	Assignment Rewind: Revise & Resubmit
2/27	Tues.	Association pt. 1: Aggregation	
2/29	Thur.	Association pt. 2: Composition	A5: Assembling the Pieces
3/5	Tues.	Inheritance	
3/7	Thur.	Exploring Inheritance in Java Classes	A6: Use What Your Parent (Class) Gave You
3/12	Tues.	Mid-semester Recap / AMA	
3/14	Thur.	Midterm Skill Check (No Class)	
3/19	Tues.	NO CLASS	
3/21	Thurs.	SPRING BREAK	
3/26	Tues.	Polymorphism pt. 1: Method Overriding	
3/28	Thur.	Polymorphism pt. 2: Method Overloading	A7: Not Your Parent’s Method
4/2	Tues.	Abstraction pt. 1: Interfaces and Generic Types	
4/4	Thur.	Abstraction pt. 2: Abstract Classes	A8: Promises, Promises…
4/9	Tues.	Final Project Workshop 1	FP1: Project Proposal
4/11	Thur.	Real World Applications pt. 1	
4/16	Tues.	Software Engineering Skill: Architecture Diagrams	FP2: Draft Codebase, Architecture Diagram
4/18	Thur.	Real World Applications pt. 2	
4/23	Tues.	Final Project: Code Review	FP3: Beta Codebase, Revised Architecture Diagram
4/25	Thur.	Real World Applications pt. 3	
4/30	Tues.	Final Project Workshop 3	FP3: Final Codebase and Documentation
5/2	Thur.	Final Project Presentations	

Late Policy

Programmers develop their skills through constant practice. CSC120 features weekly programming assignments, which build from one week to the next, and each assignment is designed to take a full week of work. Delaying the completion of an assignment will put you at a disadvantage going into the next one. Nevertheless, sometimes a little flexibility makes a huge difference, so students are empowered to give themselves extensions when it would aid their ability to master the material. Only extensions which precisely adhere to the guidelines below will be considered valid. Late assignments without a valid extension will receive no credit.

Extension Guidelines

Extensions must be requested *before* the work is due, and they will not be given retroactively. Students may only grant themselves extensions for homework assignments; extensions do not apply to lab work, skill checks, or the final project. When pair programming, both students in the pair must each submit their own request for the same extension. Work turned in under an extension will receive the lowest priority for grading, and in some cases may not be returned until the end of the semester.

Requesting an Extension

Prior to the original deadline for the assignment, submit a file called `extension.txt` on Gradescope that contains the following:

- Your name (and partner's name if you are pair programming)
- Assignment number and original due date
- Duration of extension and new due date
- Review of any prior extension requests

For example:

```
Jordan Crouser & Johanna Brewer
HW9 originally due 10/1/2021
2 day extension, now due 10/3/2021
Previous extensions: HW2 (1 day), HW7 (2 days)
```

Be sure that you **do not submit any other files** with your extension request. You can bring your code in progress to office or TA hours if you need. Do not turn in work that you do not want to be graded.

Once you've finished, and before your new deadline expires, submit all your files for the assignment as usual.

Changing an Extension

If you submit an `extension.txt` before the deadline and then decide you do not need it, just turn your code as usual and include an `extension.txt` file without any text. If you realize that you need even more time, you may grant yourself an additional extension following the same procedure again. However, repeated resets are unlikely to enhance your learning, so be mindful about how much time you add to your clock!